# Introdução a Data Science

Algoritmos de Machine Learning e métodos de análise

Casa do Código

TATIANA ESCOVEDO
ADRIANO KOSHIYAMA

# ISBN

Impresso e PDF: 978-85-7254-054-4

EPUB: 978-85-7254-055-1

MOBI: 978-85-7254-056-8


Caso você deseje submeter alguma errata ou sugestão, acesse http://erratas.casadocodigo.com.br.

# AGRADECIMENTOS

Os autores gostariam de agradecer a todos aqueles que de certa forma contribuíram para que o projeto deste livro saísse do papel e se tornasse realidade. Principalmente, à Editora Casa do Código e a Vivian Matsui, por terem nos dado a oportunidade de publicar este livro e por terem pacientemente aguardado os novos conteúdos.

## TATIANA ESCOVEDO

Aos meus pais Cristina e Mauricio, que me proporcionaram uma educação de qualidade ao longo de toda a minha vida, e a todos os professores que tive durante minha formação acadêmica, que sempre estimularam minha vontade de aprender, em especial a meus orientadores de mestrado e doutorado, prof. Carlos Lucena e prof. Marley Vellasco. Ao meu marido, prof. Rubens Melo, por ser meu companheiro de vida e de profissão, e por ser minha referência e orgulho como um dos mais importantes pesquisadores do mundo na área de Banco de Dados.

Aos meus alunos e ex-alunos da PUC-Rio, que me estimulam todos os dias a aprender mais e inventar novas maneiras para apresentar conceitos complexos de forma leve e divertida. Aos meus amigos da Petrobras, em especial ao Leonardo Matos, Carlos Rechelo, Felipe Dias, Carlos Eduardo Leal (Caê), Arthur Barbosa, Fábio Heuseler e João Carrilho, que admiro muito e que me estimulam a me tornar a cada dia uma profissional melhor. Também ao meu primeiro gerente, Zair Ramos, que sempre me incentivou desde meus primeiros anos de profissão.

Finalmente, ao meu amigo e coautor deste livro, Adriano Koshiyama, que adotei como "filho" em 2012 e que desde então é o meu maior parceiro nos trabalhos acadêmicos e profissionais de Data Science.

## ADRIANO KOSHIYAMA

À minha mãe Sayonara e ao meu pai Adriano por todos os incentivos criados e exemplos dados para a formação da pessoa que sou. Aos meus irmãos pelos momentos compartilhados, felizes e tristes, no decorrer de minha vida. Obrigado a vocês por tudo.

Aos meus orientadores de graduação, prof. Maria Cristina Lorenzon e prof. Wagner Tassinari, por me educarem em análise de dados, teoria e prática, assim como por me darem a primeira oportunidade em pesquisa acadêmica. Aos meus orientadores de mestrado, prof. Marley Vellasco e prof. Ricardo Tanscheit, por me introduzirem ao mundo da Inteligência Artificial e acreditarem no meu potencial para pesquisa, ensino e extensão. Ao meu orientador de doutorado, prof. Philip Treleaven, pela mentoria acadêmica, intelectual e pessoal.

Por fim, à minha amiga e coautora deste livro, Tatiana Escovedo, que me adotou como "mãe" e até hoje tem tomado conta de mim. Ela foi a maior incentivadora para a escrita e conclusão desta obra.

# SOBRE OS AUTORES

**Tatiana Escovedo** é Doutora em Engenharia Elétrica pela Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio, 2015) na área de Métodos de Apoio a Decisão (Machine Learning e Redes Neurais), Mestre em Informática (PUC-Rio, 2007) na área de Engenharia de Software e Bacharel em Informática (PUC-Rio, 2005). Tem experiência na área de Ciência da Computação, com ênfase em Engenharia de Software e Ciência de Dados, atuando principalmente nos seguintes temas: Desenvolvimento de Sistemas, Machine Learning, Business Intelligence e Sistemas Colaborativos. Desde 2006, é Analista de Sistemas da Petrobras e, desde 2009, é Professora e Coordenadora de cursos de pós-graduação Lato Sensu da PUC-Rio: Especialização em Análise e Projeto de Sistemas e Especialização em Ciência de Dados. Com múltiplos interesses, é apaixonada por ensinar, aprender e resolver problemas com soluções criativas. Nas horas vagas, é bailarina e pensa em maneiras de mudar o mundo. Mais informações podem ser encontradas em seu perfil no LinkedIn: https://www.linkedin.com/in/tatiana-escovedo/ e no seu currículo Lattes: http://lattes.cnpq.br/9742782503967999.

Figura 1: Tatiana Escovedo

**Adriano Koshiyama** é Doutorando em Ciência da Computação desde 2016 pela University College London (UCL), Mestre em Engenharia Elétrica pela Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio, 2014) na área de Métodos de Apoio a Decisão (Machine Learning, Estatística e Otimização) e Bacharel em Economia Pela UFRRJ (2011). Entre 2014 e 2015, foi Assistente de Pesquisa em projetos de P&D e Professor Assistente da PUC-Rio. Trabalhou como Consultor da NanoBusiness Information and Innovation na área de tecnologia e como Cientista de Dados na Sieve Price Intelligence, posteriormente adquirida pela B2W Digital SA, sendo responsável principalmente pelas estratégias de precificação automática. Entre 2016 e 2018, atuou na Nomura International PLC em seu Quant Strategies Desk (Renda Fixa) e na MindX como um cientista de dados, desenvolvendo produtos de aprendizado de máquina para avaliação psicométrica. Recentemente, ele foi estagiário no AI Labs no Goldman Sachs, trabalhando como estrategista em Machine Learning, assim como estudante associado no Alan Turing Institute. Seus principais tópicos de pesquisa estão relacionados a Ciência de Dados, Machine Learning, métodos estatísticos, otimização e finanças. Mais informações podem ser encontradas em seu perfil no LinkedIn: https://www.linkedin.com/in/koshiyama/ e no seu currículo Lattes: http://lattes.cnpq.br/3216456737953353.

Figura 2: Adriano Koshiyama

# PREFÁCIO

Quando iniciei meus estudos em Data Science (ou Ciência de Dados, em português), há muitos anos, sentia que a maioria dos livros existentes na literatura era muito complexa, recheada de demonstrações matemáticas e detalhes técnicos muitas vezes assustadores para um iniciante na área, especialmente aqueles cuja base matemática não era tão profunda.

Cada pessoa tem um estilo preferido de aprendizagem e, apesar de muitos preferirem conteúdos detalhados e profundos, o estilo que sempre funcionou para mim foi o que eu chamo de "aprendizado cebola", pois é feito em camadas: gosto de ter primeiro uma visão geral de todos os assuntos e, depois, ir me aprofundando em determinados tópicos de acordo com a minha necessidade.

Além disso, a maioria dos livros técnicos é escrita em inglês e muitos estudantes têm dificuldades de compreensão do idioma, ou mesmo preferem ler em português. Desta forma, surgiu a ideia de escrever um livro introdutório de Ciência de Dados em português, que pudesse guiar os aprendizes iniciantes nesta área fascinante, e estimulá-los a se desenvolverem em tópicos mais avançados.

Esperamos que a leitura seja agradável e que este livro possa ser útil na sua caminhada! Bons estudos.

# PÚBLICO-ALVO E ORGANIZAÇÃO DO LIVRO

Este livro é indicado para profissionais, estudantes e

professores que desejem iniciar seus estudos em Data Science e algoritmos de Machine Learning e que tenham noções de conceitos matemáticos e de lógica de programação.

O objetivo principal deste livro é mostrar como utilizar Data Science para resolver problemas e gerar produtos que agreguem valor ao negócio, aprendendo com os dados. Assim, o foco deste livro é no processo e nas técnicas relacionadas aos algoritmos preditivos mais comumente utilizados em Ciência de Dados, mas mostrando também a importância da etapa de preparação dos dados brutos, limpeza e análise exploratória.

Passaremos pelas etapas necessárias para resolução de problemas de Data Science do início ao fim, contemplando teoria e prática. Para tal, todos os conceitos teóricos apresentados serão complementados com exemplos práticos na linguagem R. O **capítulo 2** explica como instalar os requisitos necessários para trabalhar com R em seu computador e traz uma introdução aos principais comandos da linguagem.

O **capítulo 3** traz uma introdução a conceitos de Estatística e Álgebra Linear para que seja possível compreender melhor o funcionamento dos algoritmos apresentados no livro. Serão apresentados exemplos práticos com a linguagem R para melhor entendimento dos conceitos.

O **capítulo 4** aborda o tema Pré-processamento de dados, uma etapa importantíssima para o entendimento do problema e preparação dos dados para a aplicação dos algoritmos de Machine Learning, a seguir.

Os **capítulos 5 e 6** abordam problemas de Classificação, e os

**capítulos 7 e 8**, por sua vez, problemas de Regressão. Os **capítulos 9 e 10** abordam os problemas de Associação e Agrupamento, finalizando os algoritmos de Machine Learning apresentados neste livro.

Finalmente, o **capítulo 11** conclui o livro, trazendo um projeto completo de Data Science do início ao fim e sugestões de aprofundamento dos estudos.

## CÓDIGO-FONTE

Todos os códigos e bases de dados utilizados neste livro estão disponíveis no repositório do GitHub: https://github.com/tatianaesc/introdatascience.

# Sumário

**1 Introdução a Data Science** **1**

1.1 Banco de dados ou bando de dados? 1

1.2 Aplicações de Data Science 4

1.3 Dados x informação x conhecimento 7

1.4 Esquema básico de um projeto de Data Science 9

**2 Introdução a R** **13**

2.1 Comandos básicos 16

2.2 Criando estruturas de dados dentro do R 18

2.3 Trabalhando com data frames 23

**3 Conceitos básicos de estatística e álgebra linear** **32**

3.1 A matemática do Data Science 32

3.2 Conceitos básicos de Estatística 34

3.3 Conceitos básicos de Álgebra Linear 50

**4 Pré-processamento de dados** **64**

4.1 Importação de dados 65

4.2 Análise exploratória 69

4.3 Preparação e limpeza 79

**5 Modelos de Classificação** **84**

5.1 Problemas de Classificação 85

5.2 Algoritmos 93

**6 Práticas de Classificação** **141**

6.1 Árvores de Classificação 141

6.2 KNN 151

6.3 Naïve Bayes (Bayes Ingênuo) 157

6.4 Support Vector Machine (SVM) 170

**7 Modelos de Regressão** **174**

7.1 Problemas de Regressão 174

7.2 Algoritmos 179

**8 Práticas de Regressão** **196**

8.1 Regressão Linear, Árvore de Regressão e KNN para Regressão 196

8.2 Regressão Logística 206

**9 Modelos de Associação e Agrupamento** **210**

9.1 Problemas de Associação 211

9.2 Problemas de Agrupamento 223

**10 Práticas de Associação e Agrupamento** **228**

10.1 Apriori 228

10.2 K-means 240

**11 Conclusão** **247**

11.1 Sugestão de template básico para projeto de Ciência de Dados 248

11.2 Exemplo de um projeto completo usando o template 251

11.3 E agora? 272

Capítulo 1

# INTRODUÇÃO A DATA SCIENCE

## 1.1 BANCO DE DADOS OU BANDO DE DADOS?

Nosso cenário mundial atual é caracterizado pela criação e crescimento de inúmeras bases de dados, diariamente e em velocidade exponencial. Estima-se que cerca de 90% dos dados armazenados na Web tenham sido gerados nos últimos 2 anos. Para a próxima década, é prevista uma taxa de crescimento de dados mundial em torno de 40%, alcançando cerca de 44 zettabytes (44 trilhões de GB) de informações digitais em todo o mundo. Estes dados são gerados, em sua maioria, de aplicativos e web sites de redes sociais, bancos, e-commerce e sistemas internos de empresas, como ilustra a figura a seguir.

Figura 1.1: Principais fontes de dados da atualidade

Para processar e obter informação útil a partir destes dados, é

necessário automatizar diversas tarefas de coleta, processamento e análise de dados para tomada de decisão, uma vez que, devido ao grande volume de dados disponível, torna-se inviável realizar estas tarefas manualmente. Nesse contexto, surge a **Inteligência Artificial**, que visa simular o comportamento de um cérebro humano utilizando máquinas.

De forma mais técnica e de acordo com a definição do Gartner Group (https://www.gartner.com/en), **Mineração de Dados (Data Mining)** é o processo de descoberta de novas e significativas correlações, padrões e tendências em grandes volumes de dados, por meio do uso de técnicas e reconhecimento de padrões, estatística e outras ferramentas matemáticas. Para encontrar padrões, o processo de Data Mining utiliza técnicas de **Machine Learning (Aprendizado de Máquina)**.

Geralmente, Machine Learning se concentra na descoberta de padrões ou de fórmulas matemáticas que expliquem o relacionamento entre os dados, sem necessariamente se preocupar com seu grau de utilidade ou aplicação ao negócio, e estuda formas de automatização de tarefas inteligentes que seriam difíceis de serem realizadas por humanos. Já em Data Mining, o objetivo principal é a extração do conhecimento pelo analista, não levando em conta o tempo necessário ou a natureza das atividades para esta tarefa.

Assim, pode-se dizer que em Machine Learning o aprendizado é o objetivo principal, enquanto no Data Mining, o aprendizado é um meio para extração de conhecimento (que deve ser avaliado e aplicado). Também se pode dizer que Machine Learning é uma das técnicas utilizadas na área de Inteligência Artificial, pois usa

algoritmos baseados matemática e estatística para realizar a tarefa de aprendizado.

Apesar de as técnicas de reconhecimento de padrões e de análise exploratória de dados utilizadas em Data Mining e Machine Learning serem antigas e em sua grande parte provenientes da Estatística, elas só passaram a ser efetivamente usadas para exploração de dados nos últimos anos, devido a fatores como: maior volume de dados disponível, criação e popularização de Data Warehouses (grandes armazéns de dados, com arquitetura de dados voltada para a tomada de decisão), recursos computacionais potentes, forte competição empresarial e criação de diversos softwares.

O conceito de Data Science (ou Ciência de Dados), por sua vez, é mais amplo: refere-se à coleta de dados de várias fontes para fins de análise, com o objetivo de apoiar a tomada de decisões, utilizando geralmente grandes quantidades de dados, de forma sistematizada. Quase sempre, além de olhar para os dados passados para entender o comportamento dos mesmos (atividade conhecida como Business Intelligence - BI), deseja-se também realizar análises de forma preditiva, por exemplo, utilizando técnicas de Data Mining e/ou Machine Learning.

Assim, Data Science não é uma ferramenta, mas sim um conjunto de métodos com o objetivo apoiar decisões de negócio baseadas em dados. Por se tratar de uma área muito vasta, optamos por focar este livro no processo e nas técnicas relacionadas aos algoritmos preditivos mais comumente utilizados, mas mostrando também a importância da etapa de preparação dos dados brutos, limpeza e análise. Desta forma, é possível aplicar os algoritmos e

gerar um produto que agregue valor ao negócio, ou seja, aplicar a ciência para aprender com os dados, e não simplesmente executar os algoritmos. Vale a pena ressaltar, entretanto, que a literatura especializada muitas vezes utiliza os termos *Business Intelligence*, *Data Science*, *Machine Learning* e *Data Mining* de forma intercambiável e/ou complementar, muito provavelmente porque este tema tem crescido e se popularizado com grande velocidade nos últimos anos, faltando, ainda, uma melhor organização da literatura quanto à nomenclatura utilizada.

Para trabalhar com Data Science, como o nome já indica, precisamos de dados, e quanto mais dados, melhor, pois será mais fácil de encontrar os padrões ou fórmulas matemáticas que os expliquem. Estes dados podem ser oriundos de fontes (estruturadas ou não) como planilhas, documentos, imagens, bancos de dados (relacionais ou não) e Data Warehouses e, na prática, têm qualidade ruim, sendo necessário gastar um tempo considerável na sua preparação, limpeza e enriquecimento. Assim, para ser capaz de realizar todas as etapas necessárias para efetivamente gerar valor ao negócio a partir de dados brutos, consideramos que o profissional de Data Science deve ter uma formação multidisciplinar, unindo disciplinas como Estatística, Programação, Aprendizado de Máquina, Banco de Dados e Inteligência Artificial.

## 1.2 APLICAÇÕES DE DATA SCIENCE

Como aplicações de Data Science já utilizadas no mundo real, podemos citar:

- No governo dos EUA, a identificação de padrões de

transferência de fundos internacionais de lavagem de dinheiro do narcotráfico e prevenção de atentados;
- Na área de varejo, a utilização de dados do histórico de compras e preferências dos clientes para definir, por exemplo, a melhor organização de prateleiras. Isso pode facilitar as vendas casadas e a definição de hábitos de consumo e previsão de necessidades de produtos/serviços em determinada região ou para determinado público-alvo, por exemplo;
- Na área de saúde, a investigação da relação entre doenças e perfis profissionais, socioculturais, hábitos pessoais e locais de moradia, para melhor entendimento das doenças e tratamentos;
- Na área de Geologia, a identificação de litologia das rochas.

No nosso dia a dia, também é fácil perceber diversas destas aplicações como, por exemplo, sugestão de livros na Amazon.com ou de filmes no Netflix, detecção automática de e-mails spam pelo Gmail e ligações de operadoras de cartão de crédito para confirmar transações suspeitas.

Como veremos ao longo deste livro, os problemas de Data Science podem ser agrupados de acordo com suas características. As principais categorias de problemas, que serão abordados neste livro, são: *Classificação*, *Regressão*, *Agrupamento* e *Associação*. A seguir, detalharemos um pouco mais estes tipos de problemas e apresentaremos alguns exemplos.

Um típico problema de Classificação é detecção de clientes com perfis fraudulentos. Imagine a seguinte situação: um determinado cliente deseja obter um empréstimo de R$ 1000,00. O

gestor deste sistema poderia se perguntar: "Será que este cliente vai pagar o empréstimo?" ou ainda, "Qual é o melhor modelo de financiamento para este cliente (juros, prazo etc.)?". Este é um problema típico de Classificação, pois deseja-se classificar um cliente em uma das possíveis classes do problema, por exemplo, bom pagador/mau pagador ou juros/prazo/outros.

Já o problema de Regressão é parecido com o de Classificação, com a diferença de que em vez do objetivo ser determinar uma classe, tenta-se prever um valor estimado. Um exemplo de problema de Regressão é a predição do valor estimado das vendas em uma nova filial de uma determinada cadeia de lojas. Se esta pergunta for mapeada em um problema de Classificação, as respostas possíveis poderiam ser: Alto/Médio/Baixo. Se mapeada em um problema de Regressão, as respostas poderiam ser valores monetários.

Outra classe de problemas que estudaremos neste livro é o problema de Associação, que pode ser exemplificado pela oferta de novos serviços e produtos a clientes. Por exemplo, em um sistema de e-commerce, poderíamos nos perguntar: "Quem observa esse produto tem interesse em ver qual outro?" ou ainda: "Quem observa esse produto costuma comprar qual outro? ".

Finalmente, abordaremos o problema de Agrupamento (ou Clusterização), que tem o objetivo de agrupar os dados de interesse. Por exemplo, se quiséssemos determinar localidades promissoras para abertura de novas filiais de uma loja. Os bairros de uma cidade poderiam ser agrupados em localidades mais ou menos promissoras.

## 1.3 DADOS X INFORMAÇÃO X CONHECIMENTO

É importante observar que, em problemas de Data Science, o valor dos dados armazenados está diretamente ligado à capacidade de se extrair conhecimento de alto nível a partir deles, ou seja, gerar, a partir dos dados brutos, informação útil para tomada de decisão (conhecimento). Uma das etapas que mais agrega valor é o processo conhecido como **Knowledge Discovery in Databases (KDD)**, ou Descoberta de Conhecimento em Bases de Dados, que consiste em transformar dados brutos em informações úteis, gerando conhecimento para a organização. Para tal, são comumente utilizados sistemas de apoio à decisão, que auxiliam os usuários na tomada de decisão.

Segundo Fayad, KDD pode ser definido como: "Um processo de várias etapas, não trivial, interativo e iterativo, para identificação de padrões compreensíveis, válidos, novos e potencialmente úteis a partir de grandes conjuntos de dados" (FAYAD et al.1996). Esta definição mostra que a descoberta de conhecimento é um processo complexo e composto de várias etapas, que necessita do envolvimento de atores como o analista de dados (para construir modelos) e o especialista de domínio (para interpretar os resultados). Para tal, assume-se como premissa que há um conjunto de dados (também chamado de *dataset*) que pode envolver **n** atributos (também chamados de campos, colunas, variáveis), representando o hiperespaço. Quanto maior o valor de **n** e o número de registros (também chamados de linhas, transações, exemplos, instâncias), maior o conjunto de dados a ser analisado e geralmente maior é a sua complexidade.

Em KDD, pode-se dividir o processo de construção dos modelos em duas grandes etapas, que são conhecidas por diversos nomes na literatura. Costuma-se dizer que se está aprendendo - ou então treinando, construindo, formulando, induzindo um modelo de conhecimento - a partir de um conjunto de dados quando se procura por padrões nestes dados, em geral, utilizando-se um ou mais algoritmos de Data Mining. Finalmente, quando se faz uma estimativa - ou então teste, predição - dos valores desconhecidos para atributos do conjunto de dados, diz-se que o modelo está sendo aplicado.

Utilizando o exemplo que utilizamos para ilustrar problemas de classificação, suponha que temos uma base histórica de clientes e suas características como idade, renda, estado civil, valor solicitado do empréstimo e se pagou ou não o empréstimo. Esses dados são ditos *rotulados*, porque contêm a informação da classe que cada cliente pertence (bom pagador/mau pagador) e são usados para a construção do modelo. A partir do momento novos clientes solicitarem um empréstimo, podemos utilizar o modelo construído com os dados rotulados para prever se cada novo cliente será bom ou mau pagador e, assim, apoiar a decisão do gestor.

Portanto, pode-se dizer que a principal etapa do processo de KDD é o aprendizado, ou seja, a busca efetiva por conhecimentos novos e úteis a partir dos dados. É importante ressaltar que todo processo de KDD deve ser norteado por objetivos claros. Deve-se definir previamente a tarefa a ser executada e a expectativa dos conhecedores do domínio (por exemplo, taxa de erro aceitável, necessidade de o modelo ser transparente para os especialistas, natureza das variáveis envolvidas etc.). Este aprendizado pode ser

essencialmente de dois tipos: supervisionado, quando o modelo de conhecimento é construído a partir dos dados apresentados na forma de pares ordenados (entrada, saída desejada) e não supervisionado, quando não existe a informação da saída desejada e o processo de aprendizado busca identificar regularidades entre os dados a fim de agrupá-los em função das similaridades que apresentam entre si. Em problemas de Classificação e Regressão geralmente usamos aprendizado supervisionado e em problemas de Agrupamento e Associação, por sua vez, aprendizado não supervisionado.

Vale a pena ressaltar que KDD é um processo, e focar apenas nos resultados obtidos pode levar à desconsideração da complexidade da extração, organização e apresentação do conhecimento extraído. É um processo muito mais complexo que "simplesmente" a descoberta de padrões interessantes: envolve o entendimento do problema e estabelecimento dos objetivos, a negociação com os "donos" dos dados que se quer analisar, os recursos computacionais e profissionais disponíveis, a dificuldade de trabalhar com dados incompletos, inconsistentes e/ou incertos e a apresentação adequada dos resultados para os interessados.

## 1.4 ESQUEMA BÁSICO DE UM PROJETO DE DATA SCIENCE

A figura a seguir ilustra o esquema básico de um projeto de Data Science, que pode ser resumido em 7 etapas.

Figura 1.2: Esquema básico de um projeto de Data Science

Um projeto de Data Science começa com uma necessidade ou ideia. Nesta etapa inicial, deve-se primeiro ter em mente o problema que se deseja resolver, e, em seguida, os objetivos devem ser definidos, bem como elencar as perguntas que os gestores desejam responder. A segunda etapa consiste em levantar as informações necessárias e efetivamente coletar os dados para resolver os problemas levantados na etapa anterior. Estes dados são geralmente (mas não obrigatoriamente) organizados em um ou mais bancos de dados (ou Data Marts).

A partir daí, podem ser realizadas operações de ETL (*Extraction* - Extração, *Transformation* - Transformação, *Loading* - Carga) a partir dos bancos de dados, com a finalidade de prepará-los para os modelos que serão construídos futuramente. Chegamos à terceira etapa, que será melhor detalhada no capítulo 4 deste livro, onde são realizadas as atividades de pré-processamento dos dados. Essa é a etapa mais demorada e trabalhosa do projeto de Data Science, e estima-se que consuma pelo menos 70% do tempo

total do projeto. Nesta etapa, pode ser necessário remover ou complementar dados faltantes; corrigir ou amenizar dados discrepantes (*outliers*) e desbalanceamento entre classes, e selecionar as variáveis e instâncias mais adequadas para compor o(s) modelo(s) que serão construídos na etapa seguinte.

A quarta etapa consiste em elencar os modelos possíveis e passíveis para cada tipo de problema, estimar os parâmetros que compõem os modelos, baseando-se nas instâncias e variáveis pré-processadas na etapa anterior e avaliar os resultados de cada modelo, usando métricas e um processo justo de comparação. A seguir, na quinta etapa, são realizadas as atividades de pós-processamento: deve-se combinar as heurísticas de negócio com os modelos ajustados na etapa de anterior e fazer uma avaliação final, tendo em vista os pontos fortes e dificuldades encontradas na implementação de cada um dos modelos. A quarta e a quinta etapas serão detalhadas nos capítulos 5, 6, 7, 8, 9 e 10 deste livro.

Chega-se então à sexta etapa. Nela, recomenda-se relatar a metodologia adotada para endereçar a solução as demandas dos gestores; comparar os resultados do melhor modelo com o benchmark atual (caso haja) e planejar os passos para a implementação da solução de proposta. Finalmente, na sétima e última etapa, busca-se a geração de valor ao empreendimento, tanto qualitativamente (por exemplo, listar os ganhos operacionais e de recursos humanos) quanto quantitativamente (por exemplo, calculando o ROI - *Return on Investment*).

É importante ressaltar que, por mais que se fale em Data Science em temas como Machine Learning e Inteligência Artificial, o papel do ser humano é fundamental, na formulação dos

objetivos, na escolha de técnicas de pré-processamento de dados e algoritmos utilizados, na parametrização dos algoritmos para a construção dos modelos, sempre usando a sua experiência, conhecimento e intuição para tomar as melhores decisões. É por isso que se diz que o processo de Data Science é mais arte do que ciência, e não existe uma "receita de bolo" para cada um dos problemas a resolver.

Capítulo 2

# INTRODUÇÃO A R

A principal ferramenta que utilizaremos nos exemplos deste livro será a linguagem R, executada dentro do ambiente RStudio. O R é uma linguagem de programação e também um ambiente de desenvolvimento integrado para computação estatística. É um dialeto da linguagem de programação estatística **S**, que evoluiu para o pacote commercial **S++**. O R fornece uma grande variedade de técnicas estatísticas (como modelagem linear e não linear, testes estatísticos clássicos, análise de séries temporais, classificação, agrupamento, entre outros) e técnicas gráficas. Ele está disponível como software livre nos termos da Licença Pública Geral GNU e pode ser utilizado em uma grande variedade de plataformas UNIX, Windows e MacOS. Além disso, é altamente expansível, através do uso dos pacotes, que são bibliotecas para funções ou áreas de estudo específicas. Um conjunto de pacotes padrão é incluído com a instalação de R, mas existem muitos outros disponíveis na rede de distribuição do R (CRAN).

Atualmente, a linguagem R é amplamente utilizada por estatísticos e analistas de dados, e sua popularidade aumentou significativamente nos últimos anos. Existem diversas interfaces gráficas que facilitam o desenvolvimento em R, dentre as quais recomendamos a utilização do RStudio, um software livre que é um ambiente de desenvolvimento integrado para R.

O RStudio exige que você tenha instalado no computador o kit de desenvolvimento Java (JDK), que pode ser baixado a partir de http://www.oracle.com/technetwork/java/javase/downloads/index.html/. A instalação é simples e intuitiva, bastando seguir as instruções do *wizard*.

Após instalar o JDK, você deverá instalar o pacote estatístico R (https://cran.r-project.org/), além da interface de desenvolvimento RStudio (https://www.rstudio.com/products/rstudio/download/), sendo necessário seguir a ordem de instalar primeiro o R e depois o RStudio. A instalação de ambas as ferramentas também é simples e intuitiva. Recomendamos que você instale sempre a versão mais recente disponível no site.

Após realizar as instalações do JDK, R e RStudio, você já pode começar a programar em R. Para tal, será necessário abrir o RStudio e criar um novo **R Script**, através do caminho: `File->New File->R Script` . Você também pode clicar no botão com sinal de `+` e selecionar a opção `R Script` (ou, ainda, pressionando as teclas `CTRL+SHIFT+N` ). Um R Script é, essencialmente, um programa ou um trecho de programa em R. As figuras a seguir ilustram como criar um novo R Script.

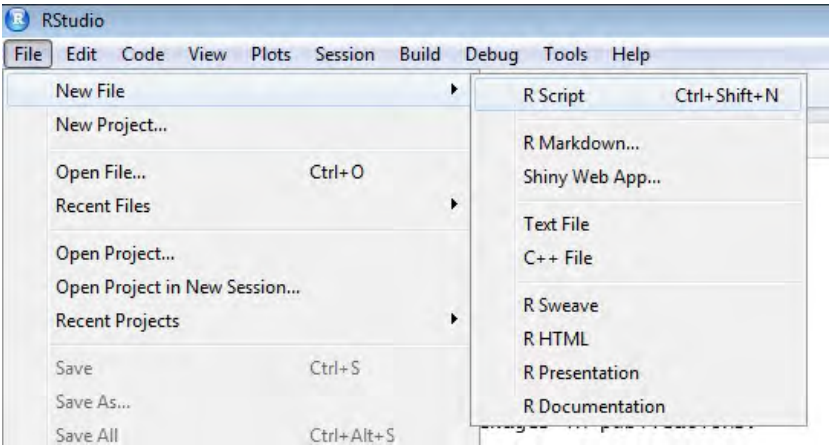



Figura 2.1: Criação de novo R Script

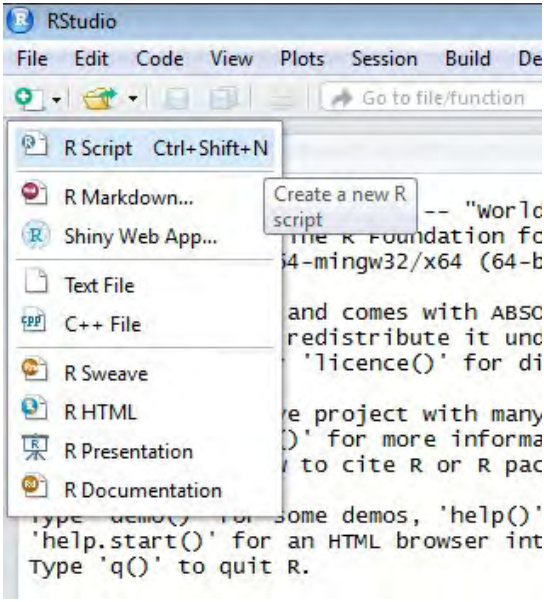



Figura 2.2: Criação de novo R Script - Alternativa

Para executar os exemplos que serão apresentados neste

capítulo e no restante do livro, após criar um novo R Script, basta digitar os comandos que serão solicitados. Em seguida, você precisará compilar e executar o código. Para tal, selecione com o cursor o bloco ou trecho com o comando e use o atalho `CTRL+Enter` ou `CTRL+R`. Você também pode selecionar o trecho de código que deseja compilar e executar e apertar botão `Run` (no topo direito do R Script).

Para salvar o seu código, basta ir a `File->Save as...` e depois dar um nome apropriado ao arquivo e salvar na pasta desejada (ou simplesmente pressionar `CTRL+S`). O arquivo será salvo no formato `.R`. Após fechar o arquivo, para reabri-lo basta selecionar `File->Open File...` e buscar o arquivo `.R` cujo nome foi dado anteriormente.

## 2.1 COMANDOS BÁSICOS

Os pacotes do R são um conjunto de funções e dados desenvolvidos pela comunidade. Eles aumentam o poder do R, pois melhoram funções nativas da linguagem ou adicionam novas. A lista dos pacotes existentes encontra-se disponível no site CRAN - *Comprehensive R Archive Network* (http://cran.r-project.org/). Quando você instala o R, já são carregados os pacotes para realizar algumas operações básicas, como as que serão demonstradas ao longo deste capítulo. Existem também os pacotes recomendados (*recommended packages*), que são instalados junto com o R, mas não são carregados quando iniciamos um programa. Por exemplo, temos os pacotes `MASS`, `lattice` e `nlme`, e, se quisermos utilizá-los, precisaremos carregá-los, com o seguinte comando:

```
> library(NOME_DO_PACOTE)
```

Quando você quiser utilizar algum outro pacote que não venha com a instalação padrão do R, você poderá instalá-lo a partir do CRAN, utilizando o seguinte comando:

```
> install.packages("NOME_DO_PACOTE")
```

Após garantir que o pacote desejado esteja instalado, para utilizá-lo, faça:

```
> library(NOME_DO_PACOTE)
```

Para verificar se algum pacote já está instalado, digite o comando:

```
> require(NOME_DO_PACOTE)
```

Se o resultado deste comando for `T`, é porque já está instalado. Se quiser consultar a documentação de algum comando, basta utilizar o comando:

```
> help("COMANDO")
```

que também pode ser escrito como:

```
> ?COMANDO
```

Nos exemplos deste livro, sempre avisaremos quando for necessário instalar um novo pacote. Outra convenção que utilizaremos é apresentar os comandos que devem ser digitados no R Script precedidos pelo sinal `<`. Você pode digitá-los no R Script (sem o `<`) e depois executar, ou pode simplesmente escrevê-los direto na console (área que fica abaixo do R Script), também sem o `<`.

## 2.2 CRIANDO ESTRUTURAS DE DADOS DENTRO DO R

Neste capítulo, vamos apresentar alguns exemplos simples em R para que você comece a se familiarizar com a linguagem e possa entender os exemplos dos próximos capítulos. Nesta seção, vamos aprender como realizar operações matemáticas e atribuições simples no R, como concatenar dados no R e como manipular as estruturas criadas. Estruturas são comumente utilizadas quando estamos trabalhando com um conjunto de dados, sendo possível incluir novos elementos, alterar ou excluir os elementos existentes. A seguir, veremos alguns exemplos de manipulação de estruturas existentes.

### Operações matemáticas e atribuições simples no R

O R antes de tudo é uma calculadora, logo algumas contas como a seguinte podem ser facilmente realizadas. Experimente executar os seguintes comandos diretamente na console:

```
> 1+1
 [1] 2
> exp(4)
 [1] 54.59815
> cos(pi)^2 + sin(pi)^2
 [1] 1
```

Note que a lógica matemática de funções aqui se mantém, isto é, `cos()` ou `sin()` é uma função que recebe um número (no caso `pi`, que o R trata como um número) e retorna o resultado dessa conta. Todas as funções têm sempre o esquema: `nomefunção(argumentos)`. Para fazer atribuições, de maneira a guardar valores para póstumas manipulações, faça:

```
> a = 1+1
> a <- 1+1
> resultado = cos(pi)^2 + sin(pi)^2
```

O objeto `a` recebe via o operador `=` ou `<-` o resultado da conta `1+1`, ao passo que o objeto `resultado` guarda o cálculo `cos(pi)^2 + sin(pi)^2`. Ao longo deste livro, vamos usar os termos: objeto, variável ou estrutura de forma indiscriminada.

## Concatenando dados no R - função c()

A função `c()` é usada para concatenar caracteres numéricos ou textuais em um vetor. Vamos experimentar criar 4 estruturas para representar a idade, o peso, a altura e o sexo de 10 pessoas. No novo R Script que você criou, digite e execute os seguintes comandos:

```
> # Dados das pessoas
> idade = c(24, 48, 32, 65, 38, 56, 74, 19, 29, 22)
> peso = c(80, 67, 49, 55, 89, 72, 45, 88, 56, 74)
> altura = c(180, 165, 162, 175, 172, 165, 168, 185, 172, 168)
> sexo = c('M','F','M','F','M','F','M','F','M','F')
```

Foram criados 4 vetores, contendo valores de idade, peso, altura e sexo de pessoas (note que o `#` é usado para inserir comentários no R, sendo o trecho após este símbolo mero comentário). Note também que não é necessário colocar o `;` no final de cada trecho de comando, de tal forma que o R omite no console a exposição dos resultados relativos ao trecho executado via `CTRL+R`.

## Inspecionando as estruturas

Podemos inspecionar essas estruturas simplesmente executando-as ou usando a função `show`:

```
> idade
 [1] 24 48 32 65 38 56 74 19 29 22
> show(idade)
 [1] 24 48 32 65 38 56 74 19 29 22
```

Para acessar um determinado elemento de uma estrutura, basta usar o operador `[i]` e dizer qual i-ésimo índice se deseja acessar no objeto. Por exemplo, caso você queira acessar o sexo da pessoa na posição 3, deve-se fazer:

```
> sexo[3]
 [1] "M"
```

Ainda, se você quiser ver os valores de altura das pessoas localizadas nas posições entre 3 e 7, faça:

```
> sexo[3:7]
 [1] "M" "F" "M" "F" "M"
```

Ou se somente quiser ver as pessoas 3 e 5, é necessário usar a função `c()` dentro do `[i]` :

```
> sexo[c(3,5)]
 [1] "M" "M"
```

## Alterando valores das estruturas e listando os objetos no ambiente

Experimente mudar o valor de algumas posições do objeto `sexo` . Para tal, basta fazer:

```
> sexo
 [1] "M" "F" "M" "F" "M" "F" "M" "F" "M" "F"
> sexo[c(3,5)] = c("F", "M")
> sexo
 [1] "M" "F" "F" "F" "M" "F" "M" "F" "M" "F"
```

Sempre que você criar um objeto, observe que ele ficará localizado no *widget* superior direito (na aba *Environment* do

RStudio). Para eliminar um objeto no qual não se deseja mais trabalhar, basta usar a função `rm()`. Experimente:

```
> a = 2+3+4+5+6+7
> rm(a)
```

Experimente agora remover de uma só vez um conjunto de objetos:

```
> a = 2+3+4+5+6+7
> b = a*2
> resultado = a+b
> rm(a, b, resultado)
```

Você pode visualizar todos os objetos criados através do comando `ls()`:

```
> ls()
 [1] "altura" "idade"  "peso"   "sexo"
```

Como você pode observar, temos 4 variáveis ou objetos no ambiente do R: peso, idade, sexo e altura. Os objetos também aparecem na aba *Environment*, no canto superior direto da tela.

## Funções simples aplicadas em um objeto

Para checar o tamanho ou número de elementos contidos no objeto use o comando `length()`:

```
> length(idade)
 [1] 10
```

E para ordená-los use o comando `sort()`:

```
> idade
 [1] 24 48 32 65 38 56 74 19 29 22
> sort(idade)
 [1] 19 22 24 29 32 38 48 56 65 74
```

Se quiser que os elementos sejam ordenados decrescentemente, use o parâmetro `decreasing = T`:

```
> sort(idade, decreasing = T)
 [1] 74 65 56 48 38 32 29 24 22 19
```

Observe que, no primeiro caso, as idades foram ordenadas do menor para o maior, ao passo que, no segundo caso, do maior para o menor.

Para visualizar as propriedades do tipo do objeto, use a função `str()`:

```
> str(idade)
 num [1:10] 24 48 32 65 38 56 74 19 29 22
> str(sexo)
 chr [1:10] "M" "F" "F" "F" "M" "F" "M" "F" "M" "F"
```

A função `str()` recebe o objeto desejado e retorna o seu tipo. O tipo de um objeto é relativo aos componentes do qual este é constituído. Por exemplo, a variável `idade` é do tipo numérico, enquanto a variável `sexo` é do tipo caractere (char). Com uma variável do tipo numérico pode-se fazer contas, ao passo que com uma do tipo char pode se fazer agrupamentos e outros tipos de manipulações.

## Operadores lógicos

- `==` (igual a)
- `>` (maior que)
- `<` (menor que)
- `<=` (menor ou igual a)
- `>=` (maior ou igual a)
- `!=` (diferente de)

Os operadores possibilitam comparar os valores de um objeto com uma ou mais referências. Por exemplo, você tenha interesse em verificar quais pessoas têm altura maior do que 170cm, deve-se fazer:

```
> altura>170
 [1]  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE
```

Retorna-se então um conjunto de `TRUE`, indicando quais são as pessoas que atendem à referência "maior que 170 (>170)" e `FALSE` as que não se comportam desta forma. Caso você queira verificar qual o sexo das pessoas maiores que 170cm, deve-se combinar o `[i]` com o operador `>170`, fazendo:

```
> sexo[altura>170]
 [1] "M" "F" "M" "F" "M"
```

Pode-se verificar que há 3 homens e 2 mulheres. Para retornar esse resultado mais bem estruturado, use a função `table()`:

```
> table(sexo[altura>170])

F M
2 3
```

## 2.3 TRABALHANDO COM DATA FRAMES

O data frame é a estrutura de dados mais famosa no R, devido à sua facilidade de manipulação e armazenamento de diferentes tipos de variáveis. Nesta seção vamos aprender a criar e manipular data frames.

### Criando um data frame

Para criar um data frame no R a partir de variáveis

preexistentes, basta executar a função `data.frame()` e ir concatenando as variáveis desejadas:

```
tabela = data.frame(altura, sexo, idade, peso)
```

A nova estrutura chamada de `tabela` guarda em seu interior as variáveis: altura, sexo, idade e peso, atributos esses de diferentes tipos e formatos. Para visualizar o data frame no RStudio use o comando:

```
View(tabela)
```

## Manipulando um data frame

As principais funções para manipulação de um data frame são:

- `str(tabela)` - apresenta os tipos e uma breve informação das variáveis que compõem o data frame tabela;
- `names(tabela)` - dispõe o nome das variáveis usado no data frame tabela;
- `head(tabela, n = 5)` e `tail(tabela, n = 5)` - mostra as 5 primeiras/últimas linhas do data frame tabela;
- `$` - usado para acessar uma determinada variável;
- `[i,j]` - trata o data frame como uma matriz, acessando a i-ésima linha e j-ésima coluna da estrutura;
- `nrow(tabela)` e `ncol(tabela)` - apresenta o número de linhas e colunas da tabela, respectivamente;
- `dim(tabela)` - apresenta as dimensões da tabela (número de linhas e colunas).

Por exemplo:

```
> str(tabela)
```

```
'data.frame':    10 obs. of  4 variables:
 $ altura: num  180 165 162 175 172 165 168 185 172 168
 $ sexo  : Factor w/ 2 levels "F","M": 2 1 1 1 2 1 2 1 2 1
 $ idade : num  24 48 32 65 38 56 74 19 29 22
 $ peso  : num  80 67 49 55 89 72 45 88 56 74
```

O código dispõe o tipo da estrutura `tabela` (data frame) composto por 10 observações de 4 variáveis. Apresenta o tipo de cada variável e algumas observações sobre elas. Observe que há um tipo que ainda não mencionamos, chamado de `Factor`. O `Factor` é uma variável categórica, ou seja, uma variável nominal composta de níveis (levels). No caso, a variável `sexo` é do tipo `Factor` (fator) composta de 2 níveis: M e F. Já a função:

```
> names(tabela)
 [1] "altura" "sexo"   "idade"  "peso"
```

revela o nome de cada variável que compõe a tabela. Para mudar o nome de alguma variável, basta usar a função combinada com o operador `[i]`:

```
> names(tabela)[1] = "Height"
> names(tabela)[1]
 [1] "Height"
> names(tabela)[1] = "altura"
> names(tabela)[1]
 [1] "altura"
```

No exemplo, trocamos o nome da variável `altura` para `Height` e depois novamente para `altura`. Esta função é bastante útil, principalmente quando importamos um data frame de uma planilha de dados, e gostaríamos de se saber qual é o nome de cada variável, assim como abreviar o nome de algumas para tornar as rotinas mais curtas ou renomear para um nome mais amigável. Já as funções:

```
> head(tabela, n = 5)
```

```
   altura sexo idade peso
 1    180    M    24   80
 2    165    F    48   67
 3    162    F    32   49
 4    175    F    65   55
 5    172    M    38   89
> tail(tabela, n = 5)
    altura sexo idade peso
 6     165    F    56   72
 7     168    M    74   45
 8     185    F    19   88
 9     172    M    29   56
 10    168    F    22   74
```

apresentam as 5 primeiras e últimas linhas da tabela, respectivamente.

O operador `$` serve para acessar um determinado atributo de um data frame. Por exemplo, caso você queira ver os valores do atributo `sexo`, faça:

```
> tabela$sexo
  [1] M F F F M F M F M F
 Levels: F M
```

Ainda, caso você não quisesse ver todos os valores, mas somente os 7 primeiros, você poderia fazer:

```
> head(tabela$sexo, 7) # note que não é necessário colocar n = 7
para funcionar
 [1] M F F F M F M
 Levels: F M
```

Já o operador `[i,j]` serve para acessar posições dentro de um data frame. Veja alguns casos:

```
> tabela[1, 1] # altura da primeira pessoa
 [1] 180
> tabela[2, 1] # altura da segunda pessoa
 [1] 165
> tabela[1, 2] # sexo da primeira pessoa
```

```
 [1] M
Levels: F M
> tabela[, 3] # dispõe as idades de todas as pessoas
 [1] 24 48 32 65 38 56 74 19 29 22
> tabela[2, ] # apresenta as informações da segunda pessoa
   altura sexo idade peso
 2    165    F    48   67
> tabela[, c(1, 3)] # apresenta somente a 1a e 3a coluna
    altura idade
 1     180    24
 2     165    48
 3     162    32
 4     175    65
 5     172    38
 6     165    56
 7     168    74
 8     185    19
 9     172    29
10    168    22
> tabela[, -2] # omite o sexo das pessoas
    altura idade peso
 1     180    24   80
 2     165    48   67
 3     162    32   49
 4     175    65   55
 5     172    38   89
 6     165    56   72
 7     168    74   45
 8     185    19   88
 9     172    29   56
 10    168    22   74
```

Note que quando usamos o operador `[i, j]` trabalhamos com um data frame pensando que ele é uma matriz. Em suma:

- Quando se omite o valor para i ou j, tornando `[, j]` ou `[i, ]` o R entende que se deseja retornar todos os registros da linha ( `[i, ]` ) ou coluna ( `[, j]` ).
- Pode-se usar a função `c()` para explicitar várias colunas desejadas.
- Quando se usa o sinal negativo, por exemplo `[, -2]` ,

indica que não se deseja retornar aquele determinado conjunto de registros, que no caso é a segunda coluna toda.

## Exportando/Importando um data frame

Para exportar o data frame para o formato `.csv`, um dos formatos mais utilizados quando trabalhamos com dados, basta escolher um nome para o arquivo (por exemplo, `BaseDados`) e usar a função:

```
> write.csv(tabela, "BaseDados.csv")
```

O arquivo `BaseDados.csv` será criado e guardará os valores presentes no data frame `tabela`.

Suponha agora que você tenha removido o data frame `tabela`:

```
> rm(tabela)
```

Contudo, você gostaria de importá-lo de volta. Para fazer isso, basta usar o comando:

```
> tabela = read.csv("BaseDados.csv")
> View(tabela)
```

Note que o R, ao importar a tabela, criou um índice para cada registro, manifestado na coluna X, como ilustra a figura a seguir:

| | X | altura | sexo | idade | peso |
|---|---|---|---|---|---|
| 1 | 1 | 180 | M | 24 | 80 |
| 2 | 2 | 165 | F | 48 | 67 |
| 3 | 3 | 162 | F | 32 | 49 |
| 4 | 4 | 175 | F | 65 | 55 |
| 5 | 5 | 172 | M | 38 | 89 |
| 6 | 6 | 165 | F | 56 | 72 |
| 7 | 7 | 168 | M | 74 | 45 |
| 8 | 8 | 185 | F | 19 | 88 |
| 9 | 9 | 172 | M | 29 | 56 |
| 10 | 10 | 168 | F | 22 | 74 |

Figura 2.3: Tabela importada

Caso se deseje remover essa variável, basta fazer:

```
> tabela = tabela[, -1]
> head(tabela) # por default, mostra as 6 primeiras linhas
   altura sexo idade peso
 1    180    M    24   80
 2    165    F    48   67
 3    162    F    32   49
 4    175    F    65   55
 5    172    M    38   89
 6    165    F    56   72
```

A tabela resultante é ilustrada pela figura a seguir:

Codigo-cap2.R × tabela ×

Filter

| | altura | sexo | idade | peso |
|---|---|---|---|---|
| 1 | 180 | M | 24 | 80 |
| 2 | 165 | F | 48 | 67 |
| 3 | 162 | F | 32 | 49 |
| 4 | 175 | F | 65 | 55 |
| 5 | 172 | M | 38 | 89 |
| 6 | 165 | F | 56 | 72 |
| 7 | 168 | M | 74 | 45 |
| 8 | 185 | F | 19 | 88 |
| 9 | 172 | M | 29 | 56 |
| 10 | 168 | F | 22 | 74 |

Figura 2.4: Tabela resultante

A partir do data frame e das variáveis que o compõe, é possível obter estatísticas descritivas e construir alguns gráficos para analisar os dados. Estes assuntos serão abordados nos próximos capítulos.

## Para saber mais:

**Alguns links úteis para aprender R**

- CRAN: http://cran.r-project.org
- Stack Overflow R section: http://stackoverflow.com/questions/tagged/r
- Quick-R: http://www.statmethods.net
- LearnR: http://learnr.wordpress.com
- R-bloggers: http://www.r-bloggers.com

Capítulo 3

# CONCEITOS BÁSICOS DE ESTATÍSTICA E ÁLGEBRA LINEAR

## 3.1 A MATEMÁTICA DO DATA SCIENCE

Muitas das pessoas interessadas em aprender Data Science e Machine Learning não possuem o conhecimento matemático necessário para obter resultados úteis no trabalho com dados. O R e outras linguagens ou ferramentas utilizadas em Data Science, como Python e Weka, já disponibilizam diversos pacotes fáceis de usar que possibilitam um certo nível de trabalho com dados mesmo para aqueles com pouco conhecimento na teoria matemática por trás dos modelos. Apesar disso, uma compreensão matemática completa de muitas dessas técnicas é necessária para entender o funcionamento interno dos algoritmos e obter bons resultados.

Há muitas razões pelas quais a matemática aplicada a Data Science é importante, como:

- Na seleção do algoritmo a ser utilizado: é necessário considerar precisão, tempo de treinamento, complexidade

do modelo, número de parâmetros e número de recursos, entre outros fatores.
- Na escolha de configurações de parâmetros e estratégias de validação.
- Na estimação do intervalo de confiança e incerteza corretos.

Todos estes tópicos serão detalhados nos próximos capítulos.

A principal questão ao estudarmos uma área interdisciplinar como Data Science é a quantidade e o nível de matemática necessários para entender suas técnicas. Não há uma resposta correta para esta pergunta, pois depende do nível e do interesse de cada indivíduo. Algumas disciplinas que podem ajudar você a trabalhar melhor com dados são Álgebra Linear, Probabilidade e Estatística, Cálculo e Complexidade de Algoritmos. O objetivo deste capítulo é introduzir alguns conceitos básicos de Álgebra Linear e Estatística, que acreditamos serem as áreas que mais contribuem para os estudos em Data Science. Caso você tenha interesse em se aprofundar nos assuntos, recomendamos que a literatura especializada seja consultada.

A Estatística é uma área muito vasta, e tem muito a dizer sobre o que acontece quando tentamos fazer alguma inferência sobre os nossos dados. Data Science, assim como a Estatística, se preocupa em *aprender a partir dos dados* e *transformar dados em informação*. A Álgebra Linear, por sua vez é conhecida por muitos como a *matemática dos dados* e estuda vetores e matrizes de números, que são a *linguagem dos dados*, e a combinação linear destas estruturas possibilita a criação de novas estruturas de dados. A Estatística utiliza a notação da Álgebra Linear para descrever os

seus métodos, assim como e os métodos de Data Science, que, além da notação da Álgebra Linear, utiliza suas ferramentas. Diversas técnicas de Data Science e Machine Learning são uma composição de técnicas provenientes da Estatística e da Álgebra Linear. Assim, para lermos e entendermos Data Science e Machine Learning, precisamos ler e entender Estatística e Álgebra Linear.

## 3.2 CONCEITOS BÁSICOS DE ESTATÍSTICA

Para ilustrar os conceitos de estatística descritiva, utilizaremos os dados de idade, peso, altura e sexo de 10 pessoas. Crie um novo *script* no RStudio e, em seguida, digite e execute os seguintes comandos:

```
> idade = c(24, 48, 32, 65, 38, 56, 74, 19, 29, 22)
> peso = c(80, 67, 49, 55, 89, 72, 45, 88, 56, 74)
> altura = c(180, 165, 162, 175, 172, 165, 168, 185, 172, 168)
> sexo = c('M','F','M','F','M','F','M','F','M','F')
> tabela = data.frame(altura, sexo, idade, peso)
```

Vamos utilizar esses dados como contexto de exemplo para as análises apresentadas a seguir. Caso você queira depois substituir esta base de dados por outra, basta alterar o nome dos argumentos que compõem cada função.

### Tipos de variáveis

Para entender melhor a base de dados em que você está trabalhando, será necessário conhecer o tipo de cada variável desta base de dados para definir as operações e inferências que poderão ser realizadas. Formalmente, uma variável é uma característica de interesse medida em cada elemento da amostra ou da população, cujos valores variam de elemento para elemento. Na prática, cada

variável será provavelmente representada por uma coluna da sua base de dados. As variáveis podem ter valores *numéricos* ou *não numéricos* e podem ser classificadas da seguinte forma:

**Variáveis quantitativas:** são as características que podem ser medidas em uma escala quantitativa, ou seja, apresentam valores numéricos que fazem sentido. Podem ser:

- **Discretas:** são mensuráveis e podem assumir apenas um número finito ou infinito contável de valores e, assim, somente fazem sentido valores inteiros. Geralmente são o resultado de contagens. Exemplos: número de filhos, número de bactérias por litro de leite, número de cigarros fumados por dia.
- **Contínuas**: características mensuráveis que assumem valores em uma escala contínua (na reta real), para as quais valores fracionais fazem sentido. Usualmente devem ser medidas através de algum instrumento. Exemplos: peso (balança), altura (régua), tempo (relógio), pressão arterial, idade.

**Variáveis qualitativas (ou categóricas):** não possuem valores quantitativos mas são definidas por várias categorias, ou seja, representam uma classificação dos indivíduos. Podem ser:

- **Nominais:** não existe ordenação dentre as categorias. Exemplos: sexo, cor dos olhos, fumante/não fumante, doente/sadio.
- **Ordinais:** existe uma ordenação entre as categorias. Exemplos: escolaridade ($1^o$, $2^o$, $3^o$ graus), estágio da doença (inicial, intermediário, terminal), mês de observação (janeiro, fevereiro, ..., dezembro).

## Estatística descritiva

Como na prática geralmente trabalhamos com bases de dados muito grandes, costuma ser inviável analisar cada um dos dados individualmente. Para nos ajudar neste problema, podemos utilizar a *estatística descritiva*, que tem o objetivo de descrever e sumarizar um conjunto de dados. As estatísticas descritivas mais comuns, que são extraídas de variáveis contínuas ou discretas, são as que buscam responder às questões:

- **Locação:** em que região do domínio da variável a amostra obtida tende a se concentrar?
- **Dispersão:** o quão espalhada a variável está em torno dessa região de concentração?
- **Associação:** qual o grau e sentido da relação entre essa variável e as demais disponíveis?

### Medidas de locação

A estatística de locação mais conhecida e utilizada é a **média aritmética**. Dado um conjunto de *n* amostras:

$$x_1, x_2, \ldots, x_n$$

A média aritmética é calculada como:

$$\bar{X} = (x_1 + x_2 + \cdots + x_n)/n$$

Para efetuar seu cálculo no R, basta usar a função `mean()` sobre uma determinada variável. Experimente digitar e executar no seu R Script o seguinte comando:

```
> mean(tabela$peso) # ou: sum(tabela$peso)/length(tabela$peso)
 [1] 67.5
```

Outra estatística bastante usada é a **mediana**, que separa um conjunto ordenado em dois subconjuntos de mesmo número de elementos. O seu uso é muito comum para desconsiderar do cálculo da locação da variável possíveis valores discrepantes (ou *outliers*) que estariam contaminando o cálculo da média aritmética. Por exemplo, considere o conjunto de dados A = {1, 2, 3, 4, 100}. Claramente, o elemento 100 é um valor discrepante dos demais, e pode ser considerado um *outlier*. Se utilizarmos a média para descrever o conjunto A, teremos o valor 22, mas se utilizarmos a mediana, teremos o valor 3, que representa melhor este conjunto. Assim, podemos perceber que, neste caso, a mediana vai representar melhor um valor típico do conjunto A, porque não é tão distorcida por valores muito altos ou muito baixos.

Para calcular a mediana devemos ordenar a amostra do menor valor para o maior. Após o ordenamento, teremos: $x_{(1)}$, $x_{(2)}$, ..., $x_{(n)}$ que são os valores ordenados, sendo portanto $x_{(1)}$ *o menor valor (mínimo) e* $x_{(n)}$ o maior valor (máximo).

Formalmente, a mediana pode ser definida como o valor maior/menor que 50% dos demais da amostra, ou ainda o valor que separa a amostra ordenada no meio. Seja $n$ o número de elementos da amostra. Se o número de elementos da amostra for ímpar, a mediana é exatamente o valor que está na posição $(n+1)/2$. Caso seja par, o valor da mediana é a média dos valores das posições $n/2$ e $(n+1)/2$. Por exemplo, experimente executar no seu *script*:

```
> n = length(tabela$altura) # 10 elementos (par)
```

```
> alturaOrdenada = sort(tabela$altura, decreasing = F) # menor pa
ra o maior
> (alturaOrdenada[(n/2)] + alturaOrdenada[(n/2 + 1)])/2 # mediana
 [1] 170
```

Assim, quando o número de elementos da série for ímpar, haverá coincidência da mediana com um dos elementos da série. Já quando este número for par, nunca haverá coincidência da mediana com um dos elementos da série: ela será sempre a média aritmética dos dois elementos centrais da série. O R dispõe de uma função chamada de `median()`, que vamos utilizar no nosso exemplo. Digite e execute no *script*:

```
> median(tabela$altura)
 [1] 170
```

Você pode estar se perguntando: que outras estatísticas de locação poderíamos obter generalizando essa ideia de maior/menor do que x%? Como visto acima, a **mediana** é um caso especial quando x% = 50%. Uma pergunta natural é: quais são os valores que são maiores do que 0%, 25%, 50%, 75% e 100% do que os demais? Esses valores têm nomes, e são chamados de:

- maior do que 0% - valor mínimo
- maior do que 25% - 1º quartil
- maior do que 50% - mediana ou 2º quartil
- maior do que 75% - 3º quartil
- maior do que 100% - valor máximo

Para calcular esses valores no R, basta usar a função `quantile()`. Digite e execute no *script*:

```
> quantile(tabela$altura)
     0%    25%    50%    75%   100%
 162.00 165.75 170.00 174.25 185.00
```

De forma mais genérica, caso se deseje computar o valor maior do que 10% da amostra, basta colocar um argumento a mais:

```
> quantile(tabela$altura, 0.10)
   10%
 164.7
```

Em Finanças, é muito comum captar esses outros valores (10%, 5% e 1% maiores do que), principalmente quando se deseja a resposta da pergunta: em 95% dos casos, qual foi a maior queda de valor de um ativo? Basta captar todas as depreciações do ativo, ordená-los e extrair o valor maior do que (1-0.95)% = 5% da amostra ordenada.

Por fim, a função `summary()` apresenta toda a região de localização de uma variável, mostrando de uma vez só todas as medidas descritivas. Digite e execute no *script*:

```
> summary(tabela$altura) # junta todas as funções min(), mean(),
median() quantile() e max() em uma só!
    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   162.0   165.8   170.0   171.2   174.2   185.0
```

Também é possível aplicar sobre um data frame a função `summary()`. Para as variáveis categóricas é apresentada a contagem dos níveis que a compõem:

```
> summary(tabela)
      altura      sexo      idade            peso
  Min.   :162.0   F:5   Min.   :19.00   Min.   :45.00
  1st Qu.:165.8   M:5   1st Qu.:25.25   1st Qu.:55.25
  Median :170.0         Median :35.00   Median :69.50
  Mean   :171.2         Mean   :40.70   Mean   :67.50
  3rd Qu.:174.2         3rd Qu.:54.00   3rd Qu.:78.50
  Max.   :185.0         Max.   :74.00   Max.   :89.00
```

**Medidas de dispersão**

A medida de dispersão mais usada, principalmente em conjunto com a média aritmética é o **desvio padrão** (s). Como o nome diz, o desvio padrão estima o nível de dispersão em torno da média aritmética. Quanto menor o desvio padrão, mais homogênea é a amostra, ou seja, mais "parecidos" são os dados. Conhecer o desvio padrão da amostra ajuda a entender se há valores discrepantes (ou *outliers*) contaminando o cálculo da média, conforme abordamos anteriormente.

Para efetuar seu cálculo, primeiro deve-se calcular a **variância** ($s^2$), dada por:

$$(x_1 - \bar{X})^2 \times (1/n) + (x_2 - \bar{X})^2 \times (1/n) + \cdots + (x_n - \bar{X})^2 \times (1/n) = s^2$$

Logo, é avaliado o desvio quadrático de cada i-ésima amostra $x_i$ em relação à média aritmética:

$$\bar{X}$$

Cada desvio quadrático é somado, ponderando-se com pesos iguais ( `1/n` ) cada um destes. O resultado deste cálculo é chamado de **variância**, que é simplesmente o desvio padrão elevado ao quadrado. Somente essa estatística já seria suficiente para medir o nível de dispersão em torno da média aritmética. Para calcular tal quantidade, deve-se usar a função no R chamada de `var()` . Experimente:

```
> var(tabela$altura) # ou ainda: mean((tabela$altura-mean(tabela$
altura))^2)
 [1] 51.73333
```

Assim, a média aritmética da altura é 171,2 cm, enquanto que a variância da altura é 51,7cm$^2$. Observe que a medida de variância é medida em cm$^2$ em vez de cm. Isso se deve à conta realizada em cada parcela, por exemplo: (170cm - 171,2cm)$^2$ x (1/10) = (1,2cm)$^2$/10 = 1,44cm$^2$/10 = 0,144cm$^2$.

Para remover o quadrado do cm é necessário aplicar a raiz quadrada (operação inversa) na variância, ou usar a função `sd()`, obtendo assim o **desvio padrão**:

$$s = \sqrt{s^2}$$

Digite e execute no *script*:

```
> sd(tabela$altura) # igual a sqrt(var(tabela$altura))
 [1] 7.192589
```

O desvio padrão da altura foi de 7,19cm. Quando analisado em conjunto com a média aritmética, sabe-se que entre a média e dois desvios padrão de distância a mais ou a menos dela, deve haver por volta de 75% da concentração da população (veja mais em http://www.mec.ita.br/~denise/teaching/MB210/Desigualdades_Markov_Chebyshev.pdf/). No nosso caso, para um desvio padrão de distância da média, isto é, entre 164cm e 178cm estão 70% dos indivíduos da amostra. E, para dois desvios padrão de distância da média, isto é, entre 157cm e 186cm estão 100% dos indivíduos da amostra.

Uma outra forma de avaliar o quão grande é o desvio padrão (o quão dispersos os dados estão em torno da média aritmética) é calculando o **coeficiente de variação** ( `%` ):

$$CV = \frac{s}{\bar{X}} \times 100$$

Experimente calcular o coeficiente de variação no seu *script*, usando o seguinte código:

```
> (sd(tabela$altura)/mean(tabela$altura))*100
 [1] 4.201278
```

Quanto menor o coeficiente de variação, menor é a dispersão de um conjunto de dados. Há sempre tentativas de classificar um coeficiente de variação como baixo, médio ou alto (veja mais em http://www.mundoeducacao.com/matematica/coeficiente-variacao.htm/). Contudo, é muito mais interessante trabalhar com o coeficiente de variação de forma comparativa, isto é, analisando coeficientes de variação de duas amostras (pessoas brasileiras e japonesas, por exemplo) de uma mesma variável (altura, peso etc.).

**Medidas de associação**

A **covariância** descreve como duas variáveis numéricas estão relacionadas. O sinal da covariância entre duas variáveis é positivo, se elas aumentam juntas, ou negativo, se elas diminuem juntas. Quando a covariância é 0, significa que as duas variáveis são completamente independentes, mas a magnitude da covariância não tem uma interpretação simples. Para facilitar a sua interpretação, geralmente ela é normalizada entre -1 e 1. Este resultado é chamado de **coeficiente de correlação de Pearson** e, quanto mais próximo de 1 ou -1, mais relacionadas estão as variáveis.

Por exemplo, se na nossa base de dados tivermos as variáveis peso, altura e IMC (índice de massa corpórea), é fácil observar que

o coeficiente de correlação de Pearson entre peso e IMC ou entre altura e IMC será muito próximo de 1, uma vez que o IMC é calculado em função do peso e da altura. Outro exemplo menos óbvio, mas no qual também se espera alta correlação, é quando medimos o número de calorias ingeridas por uma pessoa mensalmente e o ganho de peso naquele mesmo mês, pois se espera que, quanto mais calorias ingeridas, maior o ganho de peso.

No nosso exemplo, para verificar o coeficiente de correlação de Pearson entre duas variáveis usando o R, fazemos:

```
> cor(tabela$idade, tabela$peso, method = "pearson")
[1] -0.5607582
> cor(tabela$idade, tabela$altura, method = "pearson")
[1] -0.3807548
> cor(tabela$peso, tabela$altura, method = "pearson")
[1] 0.5387381
```

Calcular o coeficiente de correlação de Pearson entre duas variáveis pode ser muito útil quando queremos simplificar nossa base de dados, por exemplo, eliminando variáveis fortemente relacionadas. Neste caso, quando temos duas variáveis com alta correlação, podemos considerá-las redundantes, e pode ser interessante manter apenas uma das duas variáveis na base, diminuindo o número de variáveis trabalhadas e melhorando a performance dos modelos que serão aplicados posteriormente.

## Distribuições

A distribuição de uma variável é o padrão apresentado por um conjunto de dados resultantes da observação desta variável. Por exemplo, ao observarmos especificamente a variável `peso` de 100 indivíduos adultos de uma população, é fácil notar que o valor assumido pela variável `peso` vai variar de pessoa para pessoa. Ou

seja, os dados dos pesos das pessoas apresentam variabilidade, e o padrão desta variabilidade é a distribuição da variável. Quando representada em um gráfico, a distribuição define uma curva e a área sob esta curva determina a probabilidade de ocorrer o evento associado a ela. Existem diversos tipos de distribuição, mas aqui abordaremos as mais comuns: a distribuição normal e a lognormal.

**Distribuição normal**

A **distribuição normal** (ou Gaussiana) é a clássica curva simétrica em forma de sino. Esta é a distribuição mais popular porque muitas medições podem ser aproximadas pela distribuição normal, por exemplo, idade ou altura de uma determinada população, notas de um grupo de alunos etc. Se fizermos uma medição repetidas vezes, a tendência será resultar em uma distribuição normal. Por exemplo, se um médico pesa um paciente várias vezes, se fizer um número suficiente de medidas, terá uma distribuição normal ao redor do peso real do paciente. A variação encontrada será devido ao erro de medição. A distribuição normal é definida sobre todos os números reais.

A distribuição normal é descrita por dois parâmetros: a média $m$, que representa o centro da distribuição (pico) e o desvio padrão $s$, que representa a unidade natural de comprimento" da distribuição e permite que estimemos o quanto uma observação é rara: quanto mais distante da média (ou seja, maior o desvio padrão), mais rara ela será.

Considerando uma variável normalmente distribuída, sabe-se que:

- Aproximadamente 68% das observações estará no intervalo

*(m-s,m+s)*
- Aproximadamente 95% das observações estará no intervalo *(m-2s,m+2s)*
- Aproximadamente 99,7% das observações estará no intervalo *(m-3s,m+3s)*

Assim, uma observação que se encontra a mais de 3 desvios padrão da média pode ser considerada muito rara na maior parte das aplicações.

Vamos agora ver alguns exemplos da utilização da distribuição normal no R. Para tal, vamos definir o intervalo `[-5 5]` para gerar uniformemente 100 pontos em uma distribuição normal padrão (com média 0 e desvio padrão 1). No R, a função `dnorm(x, mean=m, sd=s)` retorna a probabilidade de se observar *x* quando retirado de uma distribuição normal com média *m* e desvio padrão *s*. Por padrão, `dnorm` assume média ( `mean` ) 0 e desvio padrão ( `sd` ) 1. Usaremos a biblioteca **ggplot2**, uma das principais bibliotecas do R para visualização de dados, para exibir o gráfico correspondente a esta distribuição normal padrão teórica.

Primeiramente, será necessário instalar a biblioteca *ggplot2* (caso você não tenha feito isso anteriormente), digitando e executando o seguinte comando no seu *R Script*:

```
> install.packages("ggplot2")
```

Em seguida, vamos gerar a distribuição normal padrão dentro do intervalo definido, e exibir o seu gráfico correspondente. Para tal, digite e execute os comandos no seu *R Script*:

```
> library(ggplot2)
> x <- seq(from=-5, to=5, length.out=100)
```

```
> f <- dnorm(x)
> ggplot(data.frame(x=x,y=f), aes(x=x,y=y)) + geom_line()
```

Você deverá visualizar um gráfico na tela similar a este:

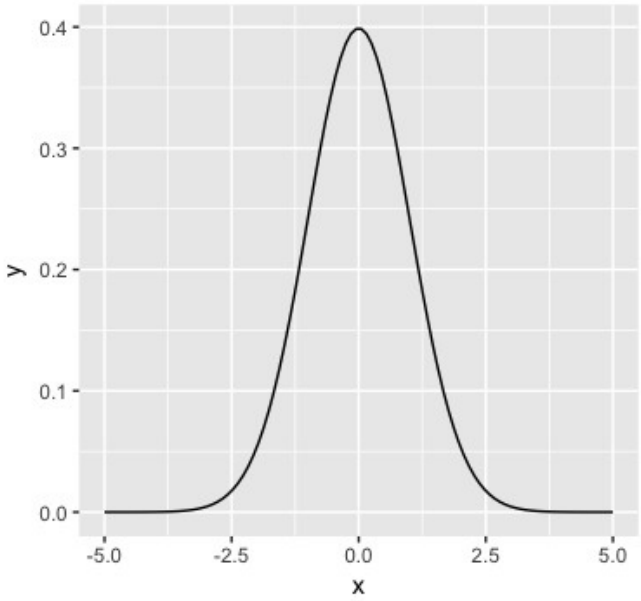



Figura 3.7: Distribuição normal padrão teórica com média 0 e desvio padrão 1. Fonte: (ZUMEL & MOUNT, 2014)

- *rDIST(n, ...)* é a função de geração aleatória de números que retorna n valores retirados da distribuição DIST.

Até o momento, trabalhamos com a distribuição normal padrão teórica, pois geramos os pontos de maneira uniforme. O que será que acontece se sortearmos aleatoriamente 1000 números a partir de uma distribuição normal padrão? Para verificar, vamos utilizar a função *rnorm(n, mean=m, sd=s)*, que gera n pontos a partir de uma distribuição com média *m* e desvio padrão *s*. Já

tínhamos guardado a distribuição gerada anteriormente (normal padrão teórica) na variável `f`. Agora, vamos gerar 1000 pontos a partir de uma normal padrão e guardar na variável `u`. Por último, vamos exibir ambas as curvas para compará-las, sendo que a primeira (normal padrão teórica) será mostrada com uma linha pontilhada e a segunda (normal padrão empírica), com uma linha cheia. O código para esta tarefa é:

```
> u <- rnorm(1000)
> ggplot(data.frame(x=u), aes(x=x)) + geom_density() +
  geom_line(data=data.frame(x=x,y=f), aes(x=x,y=y), linetype=2)
```

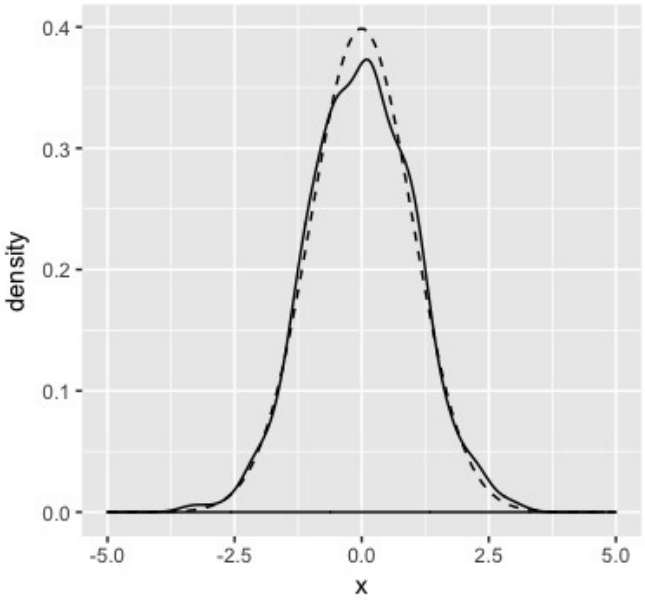



Figura 3.8: Comparação entre a normal padrão teórica e uma distribuição empírica de pontos vindos de uma distribuição normal padrão. Fonte: (ZUMEL & MOUNT, 2014)

É importante ressaltar que distribuições observadas de conjuntos de dados finitos nunca vão coincidir exatamente com as

distribuições contínuas teóricas, como a normal. A diferença entre a distribuição empírica e teórica é definida pelo tamanho da amostra utilizada.

As duas funções mostradas até agora podem ser generalizadas para outras distribuições de probabilidade. Considere uma função de distribuição de probabilidade DIST. Então:

- *dDIST(x, ...)* é a função de distribuição de probabilidade (FDP) que retorna a probabilidade de se observar o valor x.
- *rDIST(n, ...)* é a função de geração aleatória de números que retorna n valores retirados da distribuição DIST.

**Distribuição lognormal**

A **distribuição lognormal** é a distribuição de uma variável aleatória X cujo log natural *log(X)* segue uma distribuição normal. Dados positivos como salário, valor de vendas ou preços de ações geralmente podem ser modelados como uma distribuição lognormal. Esta distribuição é definida sobre todos os números reais não negativos, é assimétrica e apresenta uma longa cauda na direção de `+infinito` . Em populações que seguem uma distribuição lognormal, a média é geralmente muito maior que a mediana, devido a existência de uma pequena população com valores muito altos. Dados cujas variações são expressas como porcentagens ou diferenças relativas, e não como diferenças absolutas, são candidatos à distribuição lognormal.

O código a seguir, assim como fizemos na distribuição normal, faz a comparação entre a lognormal padrão teórica e uma distribuição empírica de pontos vindos de uma distribuição lognormal padrão, sendo que a primeira será mostrada com uma

linha pontilhada e a segunda, com uma linha cheia.

```
> x <- seq(from=-5, to=5, length.out=100)
> f <- dlnorm(x)
> u <- rlnorm(1000)
> ggplot(data.frame(x=u), aes(x=x)) + geom_density() +
  geom_line(data=data.frame(x=x,y=f), aes(x=x,y=y), linetype=2)
```

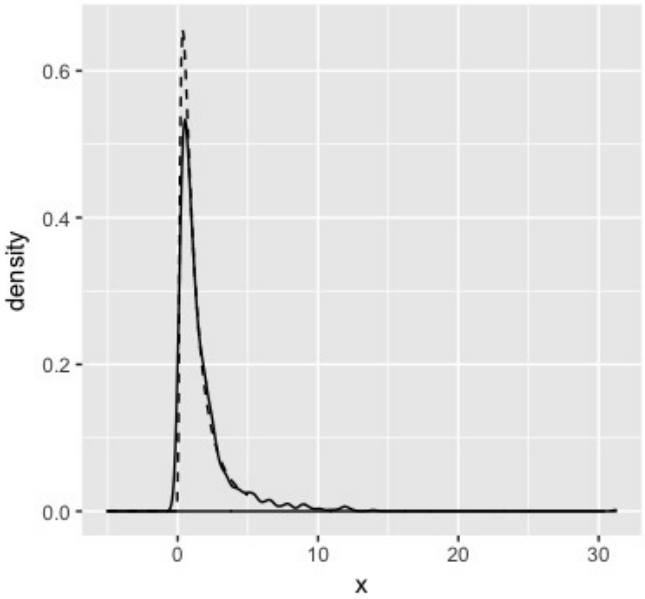



Figura 3.9: Comparação entre a lognormal padrão teórica e uma distribuição empírica de pontos vindos de uma distribuição lognormal padrão. Fonte: (ZUMEL & MOUNT, 2014)

O exemplo a seguir mostra que, se a variável aleatória X segue uma distribuição lognormal, log(X) segue uma distribuição normal. Esta distribuição empírica (linha sólida) é comparada com a normal padrão (linha pontilhada).

```
> f2 <- dnorm(x)
> u <- rlnorm(1000)
> ggplot(data.frame(u=u), aes(x=log(u))) + geom_density() +
  geom_line(data=data.frame(x=x, y=f2), aes(x=x,y=y), linetype=2)
```

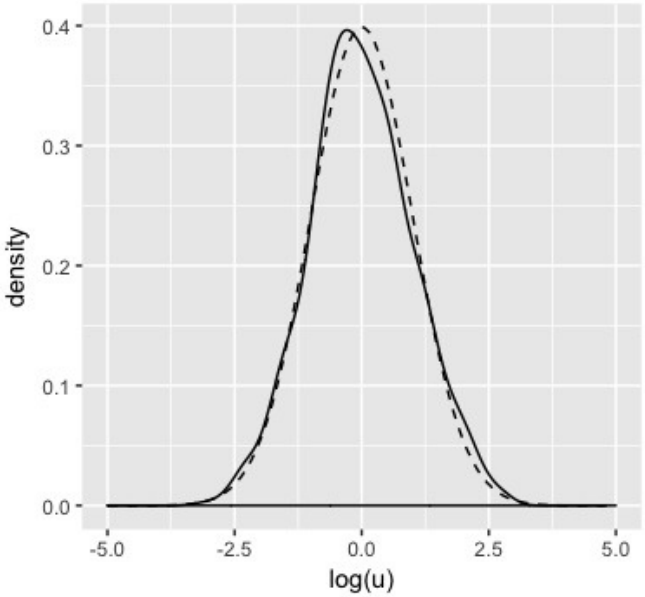



Figura 3.10: Comparação entre a normal padrão teórica e a distribuição de log(X), sendo que X segue uma distribuição lognormal. Fonte: (ZUMEL & MOUNT, 2014)

## 3.3 CONCEITOS BÁSICOS DE ÁLGEBRA LINEAR

A Álgebra Linear foi formalizada nos anos 1800 com o objetivo de encontrar soluções para equações lineares como, por exemplo:

$$y = 3 \times x + 1$$

Equações como esta são lineares porque descrevem uma linha em um gráfico bidimensional. A linha é construída atribuindo diferentes valores a $x$ para descobrir o que a equação ou modelo faz com o valor de $y$.

Considere agora o sistema de equações lineares a seguir:

$$y = 0{,}7 \times x_1 + 0{,}2 \times x_2$$
$$y = 0{,}2 \times x_1 + 0{,}5 \times x_2$$
$$y = 0{,}4 \times x_1 + 0{,}1 \times x_2$$

A coluna contendo os valores de $y$ pode ser definida como o valor de saídas da equação. De forma análoga, as duas colunas com valores inteiros são as colunas de dados, digamos $a_1$ e $a_2$, que formam a matriz $A$. Os dois valores desconhecidos $x_1$ e $x_2$ podem ser considerados os coeficientes da equação e, juntos, foram um vetor de incógnitas $b$ a serem resolvidas. Escrevendo isto de forma compacta usando Álgebra Linear, temos:

$$y = A.b$$

Problemas desta forma são geralmente difíceis de se resolver porque há mais variáveis do que equações para resolver. Além disso, muitas vezes não existe nenhuma linha que satisfaça todas as equações sem erros. E ainda, alguns problemas de Data Science que utilizam esta base teórica (por exemplo, regressão linear) podem ter um número infinito de soluções.

Além de entender a notação de Álgebra Linear, essencial para entender os algoritmos descritos nos livros e artigos, é importante entender como as operações aritméticas são executadas, tais como adição, subtração, e multiplicação de escalares, vetores e matrizes. Apesar de todas estas operações já estarem implementadas nos pacotes matemáticos das principais ferramentas de Data Science, é

necessário compreender como elas são realizadas para que seja possível efetivamente ler e escrever na notação matricial.

Você também precisará entender a decomposição de matrizes, uma técnica chave da Álgebra Linear utilizada para a redução de dados. Esta tarefa é muito comum em problemas de Data Science para reduzir dados, ou seja, trabalhar com um conjunto menor e mais simplificado dos dados do problema. A decomposição de matrizes também é uma técnica utilizada em outras operações mais complexas que veremos ao longo deste livro, como o método dos mínimos quadrados.

## Vetores

Um vetor é uma tupla de um ou mais valores denominados *escalares*, e são geralmente representados usando um caractere minúsculo, tal como *v*:

*v = (v1, v2, v3)*

onde v1, v2, v3 são valores escalares, geralmente números reais. No R, podemos criar um vetor usando:

```
> v = c(1, 2, 3)
> print(v)
 [1] 1 2 3
```

Dois vetores de mesmo tamanho podem ser multiplicados:

*c = a x b*

Assim como nas operações de adição e subtração, esta operação resulta em um vetor de mesmo tamanho:

*a x b = (a1 x b1, a2 x b2, a3 x b3)*

```
> a = c(1, 2, 3)
> b = c(4, 5, 6)
> print(a*b)
[1]  4 10 18
```

## Matrizes

Uma matriz é uma estrutura bidimensional de escalares, com uma ou mais colunas e uma ou mais linhas. Muitas vezes, em problemas de Data Science, uma matriz pode representar uma parte (ou toda) do conjunto de dados que estamos trabalhando. A notação para uma matriz é geralmente um caractere maiúsculo, tal como *A*, e suas entradas são referenciadas pelos índices de linha (i) e coluna (j), por exemplo *aij*.

*A = ((a11, a12), (a21, a22), (a31, a32))*

### Criação de matrizes

No R, uma nova matriz é criada usando a função `matrix` (o parâmetro *byrow* indica que os valores serão informados linha a linha) ou a partir de um data frame, com a função `as.matrix`. Também é possível criar uma matriz usando vetores. Experimente executar no seu script os seguintes trechos de código:

```
> A <- matrix(c(1, 0, 0, 1), nrow = 2, ncol = 2, byrow = TRUE)
> print(A)
     [,1] [,2]
[1,]    1    0
[2,]    0    1
> B <- as.matrix(data.frame(x = c(1, 0), y = c(0, 1)))
> print(B)
     x y
[1,] 1 0
[2,] 0 1
> a = c(1, 2, 3)
> b = c(4, 5, 6)
> C <- as.matrix(data.frame(a,b))
```

```
> print(C)
     a b
[1,] 1 4
[2,] 2 5
[3,] 3 6
```

**Extração de elementos**

Considerando uma matriz como o conjunto de dados que estamos trabalhando, de forma análoga a uma planilha do Excel, uma linha da matriz representa um exemplo (ou instância; por exemplo, dados referentes a um cliente bancário), e cada coluna um atributo (ou característica; por exemplo, saldo médio, número de dias no cheque especial, total de dívidas etc.). Podemos estar, então, interessados em analisar uma linha (ou cliente) em particular, uma coluna (característica) em particular, ou até mesmo um elemento (saldo médio de um determinado cliente). Desta forma, para obter um elemento específico, basta informar o índice da sua linha e o índice da sua coluna entre `[]` após o nome da matriz. Se quisermos obter uma linha inteira, basta informar a linha e deixar a coluna em branco. Se quisermos obter uma coluna inteira, basta omitir a linha e informar a coluna, assim:

```
> A <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 3, ncol = 2, byrow = TR
UE)
> A[1,2]
[1] 2
> A[1,]
[1] 1 2
> A[,1]
[1] 1 3 5
```

Usando dois pontos ( `:` ), é possível extrair várias linhas e/ou várias colunas de uma só vez:

```
> R <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12), nrow = 4,
 ncol = 3, byrow = TRUE)
```

```
> R[1:3,]
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
> R[1:3,2:3]
     [,1] [,2]
[1,]    2    3
[2,]    5    6
[3,]    8    9
`
```

### Combinação de matrizes

Imagine que você tenha dois segmentos de dados da mesma base de dados em duas matrizes diferentes e agora precisa juntá-las em uma só. É possível, então, juntar duas matrizes combinando as suas linhas usando a função `rbind()`, desde que elas tenham o mesmo número de linhas. Também é possível combinar suas colunas, usando a função `cbind()`, desde que elas tenham o mesmo número de linhas. Assim:

```
> A <- matrix(c(1, 2, 3, 4), nrow = 2, ncol = 2)
> A
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> B <- matrix(c(5, 6, 7, 8), nrow = 2, ncol = 2)
> B
     [,1] [,2]
[1,]    5    7
[2,]    6    8
> rbind(A,B)
     [,1] [,2]
[1,]    1    3
[2,]    2    4
[3,]    5    7
[4,]    6    8
> cbind(A,B)
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
```

```
[2,]    2    4    6    8
```

**Matriz Quadrada, Triangular, Diagonal e Identidade**

Existem tipos especiais de matrizes que vale a pena conhecermos. Uma matriz quadrada é aquela em que o número de linhas é igual ao número de colunas. Já a matriz triangular é um tipo de matriz quadrada em que tem todos os valores no canto superior direito ou no canto inferior esquerdo, e os demais elementos são preenchidos com zero. Por sua vez, uma matriz diagonal é aquela em que os valores da diagonal principal são diferentes de 0 e os demais valores, 0. Para construir uma matriz diagonal com 2 linhas e 2 colunas no R, faça:

```
> diag(nrow = 2, ncol = 2)
     [,1] [,2]
[1,]    1    0
[2,]    0    1
`
```

Já a matriz identidade é um tipo especial de matriz diagonal, onde os elementos da diagonal principal são todos iguais a 1 e os demais valores, 0. A notação da matriz identidade é $I_n$, onde $n$ é a ordem da matriz (número de linhas e colunas).

**Operações com matrizes**

Duas matrizes com as mesmas dimensões podem ser somadas (ou subtraídas) para criar uma terceira matriz:

$C = A + B$

Os elementos escalares da matriz resultantes são calculados como a adição dos elementos em cada uma das matrizes originais. No R:

```
> a = c(1, 2, 3)
> b = c(4, 5, 6)
> D <- as.matrix(data.frame(a,b))
> print(D)
     a b
[1,] 1 4
[2,] 2 5
[3,] 3 6
> E <- as.matrix(data.frame(b,a))
> print(E)
     b a
[1,] 4 1
[2,] 5 2
[3,] 6 3
> print(D+E)
     a b
[1,] 5 5
[2,] 7 7
[3,] 9 9
```

É possível multiplicar uma matriz por um escalar, da seguinte forma:

```
> A <- matrix(c(1, 2, 3, 4), nrow = 2, ncol = 2, byrow = TRUE)
> A
     [,1] [,2]
[1,]    1    2
[2,]    3    4
> A*2
     [,1] [,2]
[1,]    2    4
[2,]    6    8
```

Já multiplicação entre matrizes é uma operação mais complexa. Para que seja possível multiplicar duas matrizes, o número de colunas (n) da primeira matriz (A) deve ser igual ao número de linhas (m) na segunda matriz (B).

$C = A \; x \; B$

Por exemplo, se a matriz A tem *m* linhas e *n* colunas e a matriz

B tem *n* linhas e *k* colunas, o resultado será uma matriz com *m* linhas e *k* colunas:

$$C(m,k) = A(m,n) \times B(n,k)$$

De forma intuitiva, a multiplicação de matrizes consiste em calcular o produto escalar entre cada linha da matriz A e cada coluna da matriz B. Para tal, temos que multiplicar todos os elementos da primeira linha da matriz A pelos elementos da primeira coluna da matriz B e somá-los, sendo esta soma correspondente ao elemento da primeira linha e da primeira coluna da matriz resultante. Repetimos esse processo com todas as linhas e todos as colunas. No R:

```
> A <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 3, ncol = 2, byrow = TR
UE)
> print(A)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
> B <- matrix(c(1, 2, 3, 4), nrow = 2, ncol = 2, byrow = TRUE)
> print(B)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
> print(A %*% B)
     [,1] [,2]
[1,]    7   10
[2,]   15   22
[3,]   23   34
> print(B %*% A)
Error in B %*% A : argumentos não compatíveis
```

**Matriz Transposta**

A matriz transposta da matriz A é aquela que se obtém da troca de linhas por colunas da matriz A e é útil para executar operações

com os dados, por exemplo, trocando a visualização de linhas por colunas. A notação consiste em:

$$C = A^T$$

No R, basta usar a função `t()` :

```
> A <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 3, ncol = 2, byrow = TR
UE)
> print(A)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
> print(t(A))
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

**Matriz Inversa**

A matriz inversa é um tipo de matriz quadrada que possui o mesmo número de linhas e colunas que a matriz original (A). Ela ocorre quando o produto de duas matrizes (por exemplo A e B) resulta numa matriz identidade com o mesmo número de linhas e colunas. Neste caso, a matriz B é a inversa de A, ou seja:

$$A \; x \; B = B \; x \; A = I_n$$

$$B = A^{-1}$$

Há somente uma inversa para cada matriz e nem todas as matrizes são podem ser invertidas (somente quando os produtos de matrizes quadradas resultam em uma matriz identidade). Para calcular a inversa de uma matriz no R, usamos a função `solve()` . O exemplo a seguir calcula a inversa da matriz A e mostra que o produto da matriz A pela sua inversa é igual à matriz identidade:

```
> A <- matrix(c(0, 2, 1, 0), nrow = 2, ncol = 2, byrow = TRUE)
> print(A)
     [,1] [,2]
[1,]    0    2
[2,]    1    0
> print(solve(A))
     [,1] [,2]
[1,]  0.0    1
[2,]  0.5    0
> solve(A) %*% A == diag(nrow = nrow(A), ncol = ncol(A))
     [,1] [,2]
[1,] TRUE TRUE
[2,] TRUE TRUE
```

**Decomposição de matrizes**

A decomposição de uma matriz é uma forma de reduzi-la em suas partes constituintes, e tem o objetivo de simplificar operações mais complexas, que podem ser realizadas na matriz decomposta em vez de na matriz original. Uma analogia para entender a decomposição de matrizes é a fatoração de números, ou seja 10 = 5 x 5. Assim como na fatoração, há várias técnicas para a decomposição de matrizes. A seguir, apresentaremos de forma muito resumida algumas técnicas de decomposição utilizadas.

**Decomposição LU**

A decomposição LU vem do inglês *lower* e *upper* e decompõe uma matriz não como o produto de uma matriz triangular inferior (lower) e uma matriz triangular superior (upper):

$$A = L \, . \, U$$

Onde A é a matriz quadrada que queremos decompor, L é a matriz triangular inferior e U a matriz triangular superior. Esta técnica é muito usada para simplificar a resolução de sistemas de equações lineares e para encontrar os coeficientes em uma

regressão linear.

A decomposição LUP é uma variação da decomposição LU e consiste em:

$$A = L \,.\, U \,.\, P$$

Na decomposição LUP, as linhas da matriz são reordenadas para simplificar o processo de decomposição e a matriz P adicional especifica uma maneira de permutar o resultado ou retornar o resultado para a ordem original. Experimente no R:

```
> library(Matrix)
> A <- matrix( c ( 1, 2, 2, 1 ), nrow=2, byrow=TRUE)
> luA = lu(A)
> expand(luA)
$L
2 x 2 Matrix of class "dtrMatrix" (unitriangular)
     [,1] [,2]
[1,]  1.0    .
[2,]  0.5  1.0

$U
2 x 2 Matrix of class "dtrMatrix"
     [,1] [,2]
[1,]  2.0  1.0
[2,]    .  1.5

$P
2 x 2 sparse Matrix of class "pMatrix"

[1,] . |
[2,] | .
```

**Decomposição SVD**

A decomposição SVD (*Singular-Value Decomposition*) é uma técnica utilizada em diversos algoritmos de Data Science para simplificar dados, por exemplo, fornecendo uma melhor

aproximação linear ou reduzindo a dimensão dos dados. A decomposição SVD decompõe uma matriz A em:

$$A = U \,.\, \Sigma \,.\, V^T$$

onde A é a matriz m x n que se deseja decompor, U é uma matriz m x m, Σ é uma matriz diagonal m x n e $V^T$ é a uma matriz n x n, transposta da matriz V. No R, o exemplo é ainda mais simples:

```
> A <- matrix( c ( 1, 2, 2, 1 ), nrow=2, byrow=TRUE)
> svd(A)
$d
[1] 3 1

$u
           [,1]       [,2]
[1,] -0.7071068 -0.7071068
[2,] -0.7071068  0.7071068

$v
           [,1]       [,2]
[1,] -0.7071068  0.7071068
[2,] -0.7071068 -0.7071068
```

## Para saber mais:

**Livros para aprofundar seus estudos em estatística**

- MONTGOMERY, Douglas C.; RUNGER, George C. *Estatística Aplicada e Probabilidade para Engenheiros*. 5a Edição; Rio de Janeiro: LTC, 2012.
- ANTON, H. ; RORRES, C. Álgebra Linear com Aplicações; Porto Alegre: Bookman, 2004.

## Referências bibliográficas do capítulo:

ZUMEL, Nina; MOUNT, John. *Practical data science with R*. Manning Publications Co., 2014.

Capítulo 4

# PRÉ-PROCESSAMENTO DE DADOS

Muitos dos projetos de Data Science iniciam quando alguém lhe apresenta uma pilha de dados e pede que seja extraído algum significado deles. É possível utilizar o Excel para organizá-los, mas esta é uma tarefa que toma muito tempo. Utilizar o R para manipular dados é fácil porque a principal estrutura de dados (data frame) é ideal para trabalhar com dados estruturados, e o R tem adaptadores que leem dados dos principais formatos usuais, tais como `.xlsx`, `.csv`, JSON, e XML.

Relembrando o esquema básico de um projeto de Data Science apresentado no capítulo 1:

Figura 4.1: Esquema básico de um projeto de Data Science

O foco deste capítulo é na etapa de **Pré-processamento de dados** e mostrará, resumidamente, como importar uma base de dados para o R, analisá-los de forma a compreendê-los melhor e realizar as etapas de preparação e limpeza, para que os dados estejam prontos para a aplicação dos modelos de Machine Learning, que serão apresentados nos próximos capítulos. Se você está lendo este livro na edição impressa ou em um e-reader sem cores, é possível ver as imagens originais coloridas em https://github.com/tatianaesc/introdatascience/.

## 4.1 IMPORTAÇÃO DE DADOS

Uma das formas mais comuns de armazenar dados em uma base é através do formato `.csv` (*Comma Separated Value*), que consiste em dados estruturados em um formato de tabela com cabeçalho: os dados são organizados em linhas e colunas, e primeira linha traz os nomes das colunas. Cada coluna representa um fator ou medida diferente e cada linha representa uma instância ou exemplo. Há muita disponibilidade de dados públicos

em `.csv`. A figura a seguir ilustra este formato:

```
sepal_length,sepal_width,petal_length,petal_width,species
5.1,3.5,1.4,0.2,setosa
4.9,3.0,1.4,0.2,setosa
4.7,3.2,1.3,0.2,setosa
4.6,3.1,1.5,0.2,setosa
5.0,3.6,1.4,0.2,setosa
5.4,3.9,1.7,0.4,setosa
4.6,3.4,1.4,0.3,setosa
5.0,3.4,1.5,0.2,setosa
```

Figura 4.2: Exemplo de dados no formato .csv

Abrindo um arquivo `.csv` no Excel, você verá algo como:

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | sepal_length | sepal_width | petal_length | petal_width | species |
| 2 | 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 3 | 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 4 | 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 5 | 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| 6 | 5.0 | 3.6 | 1.4 | 0.2 | setosa |
| 7 | 5.4 | 3.9 | 1.7 | 0.4 | setosa |
| 8 | 4.6 | 3.4 | 1.4 | 0.3 | setosa |
| 9 | 5.0 | 3.4 | 1.5 | 0.2 | setosa |
| 10 | 4.4 | 2.9 | 1.4 | 0.2 | setosa |

Figura 4.3: Exemplo de dados no formato .csv no Excel

Nesta base de dados, cada linha representa uma observação de uma flor, e as colunas representam as características de cada flor. A maioria das colunas são medidas numéricas (largura e comprimento da sépala e da pétala) e a última coluna é o tipo de flor. Esta base de dados, conhecida como Iris, será utilizada para os exemplos deste capítulo.

A base de dados em formato `.csv` pode estar disponível online ou através de um arquivo. Você precisará importar estes

dados, armazenando-os em um data frame do R, para poder trabalhar com eles. Experimente importar os dados do dataset Iris, que estão disponíveis online. Para executar esta parte prática, copie os comandos a seguir no seu R Script e execute (ou baixe o script diretamente de https://github.com/tatianaesc/introdatascience).

```
> dados <- read.table(
>   'https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv',
>   sep=',', header=T
> )
```

Como você pode ver, a função `read.table` recebe 3 parâmetros: a URL de onde os dados serão importados, o caractere separador de colunas (no caso, vírgula) e o indicativo se há cabeçalho na primeira linha. Após executar este comando, você poderá verificar que os dados realmente foram importados para o R, pois é possível observar a criação do dataset *dados*, no **Global Environment**, como mostra a figura a seguir:

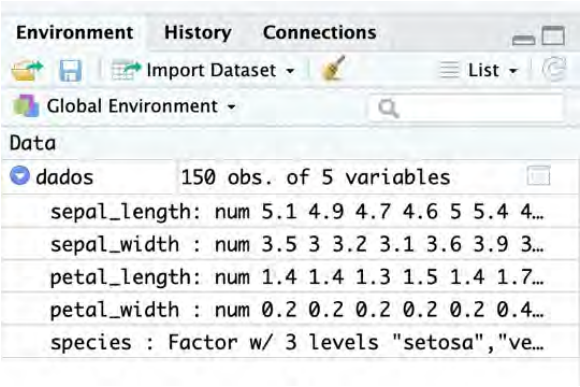

Environment
History
Connections
Import Dataset
List
Global Environment
Data
dados
150 obs. of 5 variables
sepal_length: num 5.1 4.9 4.7 4.6 5 5.4 4…
sepal_width : num 3.5 3 3.2 3.1 3.6 3.9 3…
petal_length: num 1.4 1.4 1.3 1.5 1.4 1.7…
petal_width : num 0.2 0.2 0.2 0.2 0.2 0.4…
species : Factor w/ 3 levels "setosa","ve…

Figura 4.4: Global Environment

Agora, vamos realizar algumas operações sobre este data

frame. Para verificar o tipo do objeto `dados`, digite o comando:

```
> class(dados)
[1] "data.frame"
```

Para exibir a distribuição de cada variável do data frame, digite o comando:

```
> summary(dados)
  sepal_length    sepal_width     petal_length    petal_width
       species
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
setosa    :50
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
versicolor:50
 Median :5.800   Median :3.000   Median :4.350   Median :1.300
virginica :50
 Mean   :5.843   Mean   :3.054   Mean   :3.759   Mean   :1.199

 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800

 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

Para verificar quantas linhas e colunas há nos dados, faça:

```
> dim(dados)
[1] 150   5
```

O comando `read.table()` aceita diferentes separadores e pode ler de arquivos locais ou URLs remotas. Caso quiséssemos importar os dados de um arquivo `.csv` local, (o que é mais comum) em vez de uma URL remota, poderíamos fazer:

```
> dados2 <- read.table("iris.csv", sep=',', header=T)
```

Vale a pena observar que esta base de dados com que estamos trabalhando é nativa do R, então, para trabalhar com ela, não seria necessário importar da Web, nem de um arquivo local. Bastaria executar o comando:

```
> dados3 <- iris
```

É importante ressaltar que os dados nem sempre estão prontos para uso, e muitas vezes será necessário consultar o dicionário de dados da documentação que os acompanha para entendê-los melhor e prepará-los. Fazer a preparação pelo R, e não manualmente, é uma boa prática.

## 4.2 ANÁLISE EXPLORATÓRIA

É muito importante compreender bem os seus dados a fim de obter os melhores resultados possíveis nos algoritmos de Machine Learning, aplicados posteriormente. Para aprender mais sobre seus dados de forma mais fácil e intuitiva, é possível usar a visualização de dados, ou seja, a criação de gráficos a partir dos dados brutos. Após entendê-los, você poderá limpar, transformar e apresentar melhor os dados que você possui, e posteriormente, aplicar os algoritmos de Machine Learning mais adequados para eles.

Através dos gráficos criados, será possível identificar valores discrepantes, faltantes ou inválidos, além se sugerir a necessidade da aplicação de alguma transformação de dados. Examinar visualmente os relacionamentos entre os atributos pode ajudar a identificar atributos redundantes, por exemplo.

Vamos então criar alguns gráficos com a base de dados que importamos anteriormente. Primeiro, vamos exibir os dados no formato de histograma, que é um gráfico de barras de um atributo numérico dividido em compartimentos com a altura, que mostra o número de instâncias que se enquadram em cada compartimento. Os histogramas são muito utilizados para obter uma indicação da distribuição de um atributo.

Para criar um histograma da nossa base de dados, digite os comandos:

```
> par(mfrow=c(1,4))
> for(i in 1:4) {
>  hist(dados[,i], main=names(dados)[i])
> }
```

Você notará que o seguinte gráfico será exibido na área de **Plots** do R:

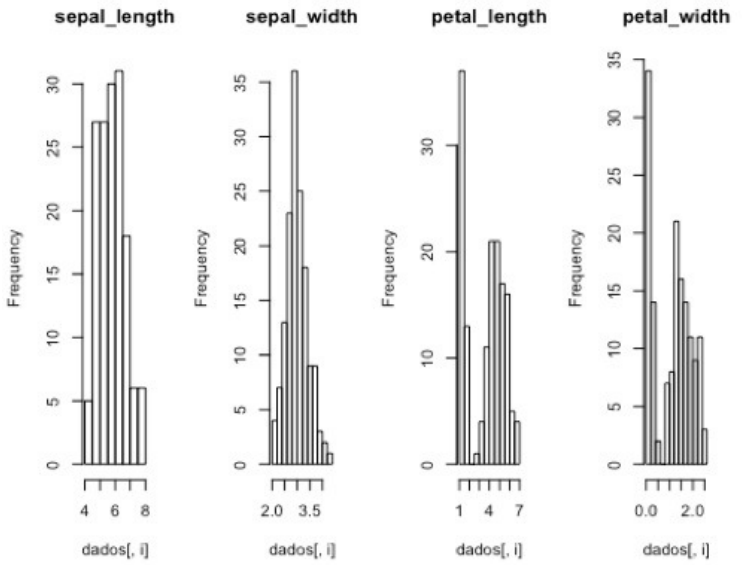



Figura 4.5: Histogramas

Analisando os histogramas criados, podemos perceber que a maioria dos atributos mostra uma distribuição normal (já detalhada no capítulo 3) ou normal multimodal (combinação de mais de uma distribuição normal).

Vamos agora criar gráficos de densidade, para ter uma melhor ideia da distribuição de cada variável. Para criar este tipo de gráfico, será necessário carregar o pacote **lattice**:

```
> library(lattice)
```

A seguir, os gráficos de densidade podem ser criados:

```
> par(mfrow=c(1,4))
> for(i in 1:4) {
>  plot(density(dados[,i]), main=names(dados)[i])
> }
```

O resultado que você obterá da execução desses comandos pode ser visualizado na figura a seguir:

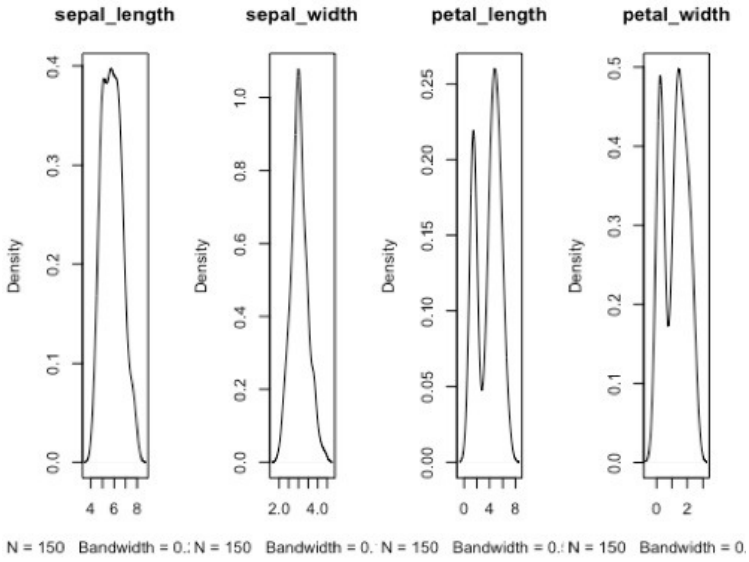



Figura 4.6: Gráficos de densidade

Outra forma interessante de visualizar os dados é através de

**boxplots**, um tipo de gráfico usado para avaliar a distribuição empírica dos dados. Para tal, faça:

```
> par(mfrow=c(1,4))
> for(i in 1:4) {
>   boxplot(dados[,i], main=names(dados)[i])
> }
```

O gráfico resultante será:

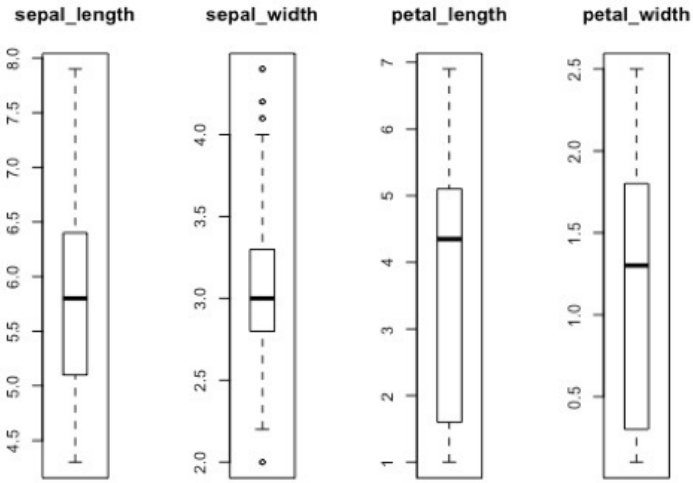



Figura 4.7: Boxplots

Nesse gráfico, dentro do quadrado, encontramos 50% dos dados, delimitados pelo primeiro e terceiro quartil e pela mediana. Fora do quadrado, os delimitadores representados por pequenas linhas retas mostram o limite razoável dos dados, e os pequenos círculos fora destes delimitadores representam possíveis *outliers*.

Uma forma interessante de analisar dados é buscar por algum tipo de relação entre os atributos usando, por exemplo, o gráfico de correlação. Este gráfico é formado a partir do cálculo da correlação entre cada par de atributos numéricos, e estas correlações podem ser plotadas em uma matriz de correlação para dar uma ideia de quais atributos variam juntos.

Para criar o gráfico de correlação, você precisará primeiro instalar e carregar o pacote **corrplot:**

```
> install.packages("corrplot")
> library(corrplot)
```

A seguir, calculamos as correlações e criamos o gráfico de correlação:

```
> correlations <- cor(dados[,1:4])
> corrplot(correlations, method="circle")
```

O resultado exibido será o da figura a seguir:

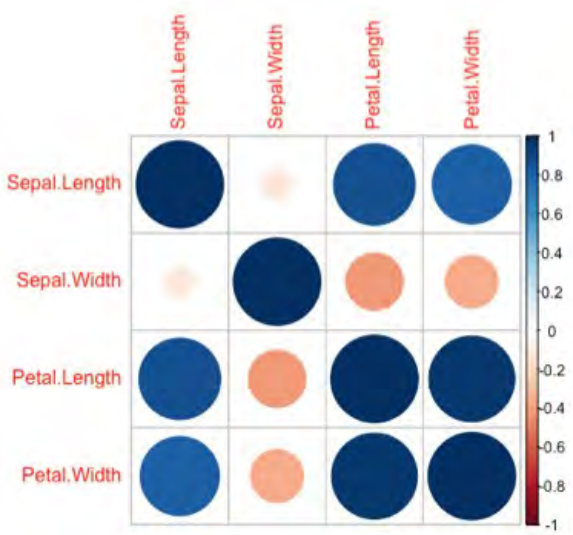



Figura 4.8: Gráfico de correlação

Nessa figura, a cor azul representa correlação positiva e a cor vermelha correlação negativa, e quanto maior o círculo, maior a correlação. É possível perceber que a matriz é simétrica e que os atributos diagonais são perfeitamente correlacionados positivamente, uma vez que mostram a correlação de cada atributo consigo mesmo. Nota-se que alguns dos atributos são altamente correlacionados.

O gráfico de dispersão, por sua vez, representa duas variáveis juntas, uma em cada um dos eixos **x** e **y** com pontos mostrando a sua interação. A propagação dos pontos indica a relação entre os atributos. Vamos criar, de uma só vez, um gráfico de dispersão para cada um dos pares de atributos do dataset. Basta executar o comando:

```
> pairs(dados)
```

Note que a matriz resultante é simétrica, e exibe os mesmos gráficos com os eixos invertidos, possibilitando analisar os dados de várias perspectivas. É possível notar uma relação linear, representada pela linha diagonal, entre o comprimento e a largura da pétala. O resultado obtido será o seguinte:

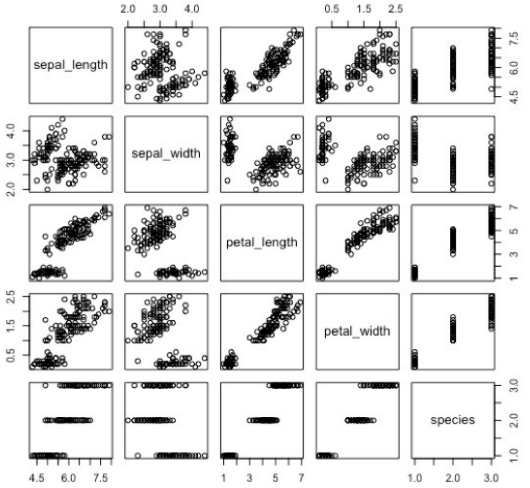



Figura 4.9: Gráfico de dispersão

Em problemas de classificação, pode ser útil colorir os pontos do gráfico de dispersão pela classe para ter uma ideia da facilidade da separação das classes do problema. Para tal, faça:

```
> pairs(species~., data=dados, col=dados$species)
```

O resultado será o da figura a seguir, que mostra uma clara separação dos pontos na maioria dos gráficos. As cores verde, vermelho e preto representam cada uma das 3 classes do problema:

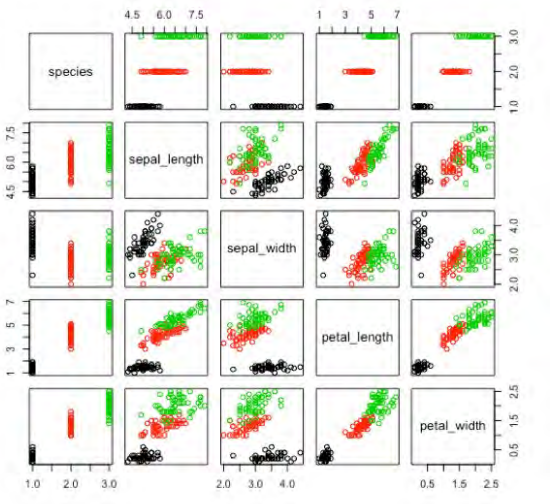



Figura 4.10: Gráfico de dispersão colorido

Assim como no gráfico de dispersão, os gráficos de densidade e *boxplot* também podem ser separados por classe. Para isso, será necessário instalar e carregar o pacote **caret**:

```
> install.packages("caret")
> library(caret)
```

Em seguida, crie um gráfico de densidade para cada atributo, separado por classe, executando os comandos:

```
> x <- dados[,1:4]
> y <- dados[,5]
> escalas <- list(x=list(relation="free"), y=list(relation="free"
))
> featurePlot(x=x, y=y, plot="density", scales=escalas)
```

O resultado será o seguinte:

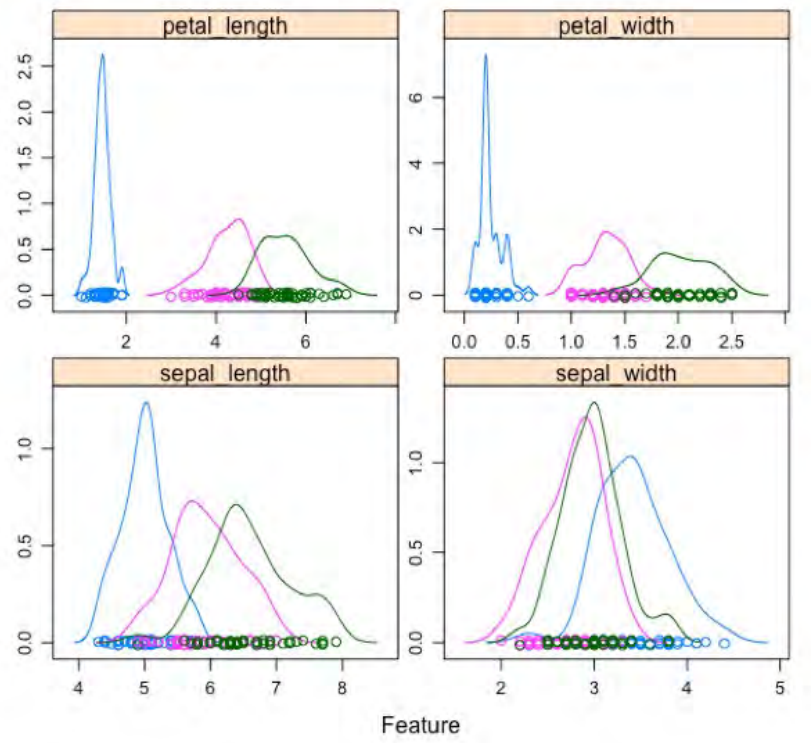



Figura 4.11: Gráficos de densidade separados por classe

Agora crie um *boxplot* para cada atributo, separado por classe, executando o comando:

```
> featurePlot(x=x, y=y, plot="box)
```

O resultado será:

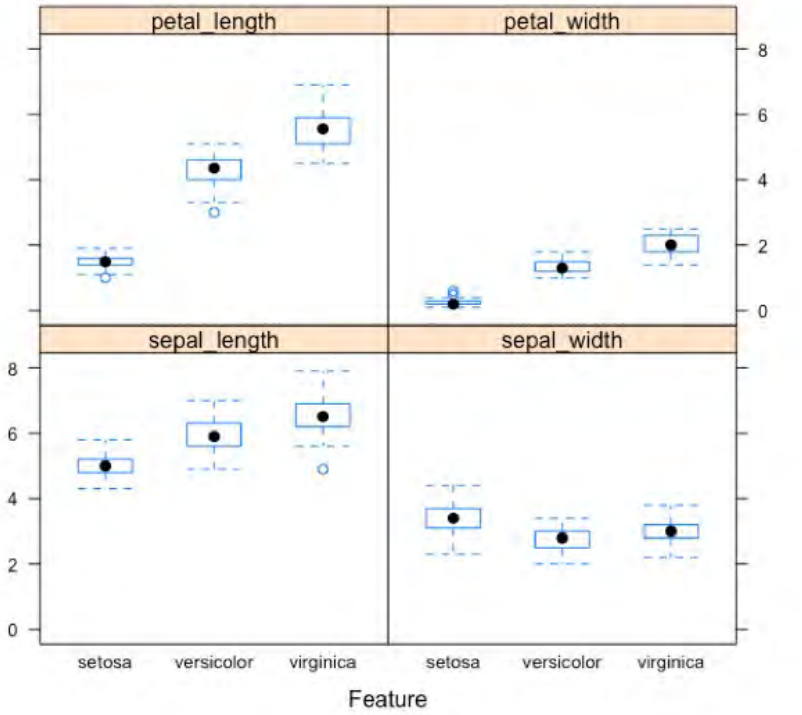



Figura 4.12: Boxplots separados por classe

Os exemplos deste capítulo ilustram algumas das possíveis formas de exploração de dados, mas é importante ressaltar que eles não são exaustivos. Não existe uma receita pronta para o trabalho de análise exploratória de dados, uma vez que ele dependerá dos dados a serem analisados e dos *insights* que cada etapa de análise provocará. Também vale destacar que existem diversos pacotes no R para criação de gráficos, além dos exemplificados neste capítulo. Um dos mais usados é o pacote **ggplot2** (consulte https://ggplot2.tidyverse.org/ para mais detalhes).

## 4.3 PREPARAÇÃO E LIMPEZA

Após examinar com cuidado os seus dados, você pode identificar alguns problemas que devem ser tratados nesta etapa, tais como:

- **Unidades de medida**: é importante sempre observar a unidade das variáveis (exemplo: idade medida em meses ou anos, renda em dólares ou reais, distância em quilômetros ou milhas etc.) para fazer as devidas conversões, se necessário.

- **Valores faltantes**: se um atributo tem muitos registros faltantes, ele não deve ser utilizado como entrada de um modelo sem um tratamento apropriado. No R, muitas das implementações de algoritmos ignoram silenciosamente por padrão linhas com valores faltantes.

- **Valores inconsistentes**: é importante entender se os valores inconsistentes são valores inválidos ou **outliers**. Um exemplo de valor inválido seria encontrar valores negativos em um campo que só pode ser positivo, ou um texto quando se espera um número. Já **outliers** são valores fora do intervalo esperado.

- **Intervalo dos valores**: preste atenção no intervalo dos valores das variáveis. Atributos como salário podem conter valores de 0 a mais de meio milhão de dólares, por exemplo. Este grande intervalo pode ser um problema para alguns modelos, e pode ser necessário transformar esta coluna usando uma transformação logarítmica para reduzir o seu intervalo de valores. Já variáveis com valores em

intervalos pequenos (exemplo: idades entre 50 e 55) provavelmente não serão uma informação útil para os modelos. Vale lembrar de que o intervalo de valores adequado varia de acordo com o domínio da aplicação.

Para exemplificar o tratamento destes problemas, vamos agora trabalhar com uma base de dados diferente e mais complexa do que a base de dados Iris (que não apresenta muitos problemas relevantes). Vamos trabalhar com o dataset do banco de crédito alemão (http://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29/), que consiste em dados tabulares sem cabeçalho, que usa uma codificação de valores explicada na documentação do dataset.

Para executar esta parte da prática, coloque o arquivo `custdata.tsv`, disponível em https://github.com/tatianaesc/introdatascience, no mesmo diretório do seu R Script. Copie os comandos a seguir no seu R Script e execute (ou baixe o script diretamente de https://github.com/tatianaesc/introdatascience).

Vamos então carregar os novos dados no data frame `dados4`, e examiná-los com o comando `summary`:

```
> dados4 <- read.table('custdata.tsv',header = T,sep = '\t')
> summary(dados4)
     custid        sex     is.employed         income
       marital.stat health.ins
 Min.   :   2068   F:440   Mode :logical   Min.   : -8700   Divor
ced/Separated:155   Mode :logical
 1st Qu.: 345667   M:560   FALSE:73        1st Qu.: 14600   Marri
ed           :516   FALSE:159
 Median : 693403           TRUE :599       Median : 35000   Never
 Married      :233   TRUE :841
```

```
 Mean   : 698500           NA's :328       Mean   : 53505   Widow
ed           : 96
 3rd Qu.:1044606                           3rd Qu.: 67000

 Max.   :1414286                           Max.   :615000


                     housing.type recent.move      num.vehicles
       age              state.of.res
 Homeowner free and clear    :157   Mode :logical   Min.   :0.000
   Min.   :  0.0   California  :100
 Homeowner with mortgage/loan:412   FALSE:820       1st Qu.:1.000
   1st Qu.: 38.0   New York    : 71
 Occupied with no rent       : 11   TRUE :124       Median :2.000
   Median : 50.0   Pennsylvania: 70
 Rented                      :364   NA's :56        Mean   :1.916
   Mean   : 51.7   Texas       : 56
 NA's                        : 56                   3rd Qu.:2.000
   3rd Qu.: 64.0   Michigan    : 52
                                                   Max.   :6.000
   Max.   :146.7   Ohio        : 51
                                                   NA's   :56
                   (Other)     :600
```

Analisando os resultados, podemos perceber que:

- A variável `is.employed` tem valores faltantes para aproximadamente 1/3 dos dados;
- A variável `income` tem valores negativos, potencialmente inválidos;
- Segundo a variável `health.ins`, cerca de 84% dos clientes têm seguro de saúde;
- Há 56 registros faltantes nas variáveis `housing.type`, `recente.move` e `numvehicles`;
- A média da variável `age` parece fazer sentido, mas os valores mínimo e máximo parecem improváveis.

Quanto ao problema de valores faltantes, no caso das variáveis `housing.type`, `recente.move` e `numvehicles`, como são

poucas linhas com valores faltantes e a mesma quantidade para as 3 variáveis, podemos simplesmente excluir estas 56 linhas da base de dados. Caso você suspeite que os `NAs` são devido a alguma falha na coleta de dados, você também pode substituí-los por algum valor que faça sentido, por exemplo, pela média dos valores.

Observando os valores inconsistentes, valores negativos de renda podem significar erro na entrada de dados, ou que o cliente tem um débito. É preciso decidir: você pode apagar estas linhas ou converter os valores negativos para zero. Além disso, podemos inferir que as idades 0 e maiores que 110 são *outliers*, podendo significar erro na entrada de dados, porém, a idade zero pode significar idade desconhecida ou que a pessoa se recusou a informar, e há a chance de que alguns clientes tenham idade avançada.

A normalização é um recurso útil quando as quantidades são menos significativas que os valores relativos. Por exemplo, podemos estar menos interessados na idade absoluta do cliente do que o quanto mais velho ou mais novo ele é em relação a um cliente típico. Neste caso, a idade dos clientes pode ser normalizada pela média das idades: quanto maior o valor da variável normalizada, mais velho é o cliente. Para tal, execute os seguintes comandos:

```
> summary(dados4$age)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    0.0    38.0    50.0    51.7    64.0   146.7
> idadeMedia <- mean(dados4$age)
> dados4$age.normalized <- dados4$age/idadeMedia
> summary(dados4$age.normalized)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.7350  0.9671  1.0000  1.2379  2.8372
```

## Resumo

Este capítulo apresentou, de forma resumida, algumas técnicas de pré-processamento de dados, como análise exploratória, preparação e limpeza. Finalizada esta etapa, podemos aplicar os modelos de Machine Learning de acordo com o problema que queremos resolver.

Capítulo 5

# MODELOS DE CLASSIFICAÇÃO

Conforme apresentamos no capítulo 1, os problemas de Data Science podem ser agrupados de acordo com suas características. Quando se deseja fazer a predição de um valor, o problema pode ser de *Classificação* (quando a variável a ser predita é categórica, geralmente uma classe) ou de *Regressão* (quando se deseja fazer a predição de um valor numérico, contínuo ou discreto). Em um problema de Classificação, poderíamos perguntar: "Devo **conceder ou não** crédito para um cliente?", enquanto em um problema de Regressão a pergunta seria: "Devo conceder **qual** valor de crédito para um cliente?". Este e o próximo capítulo detalharão os problemas de Classificação, e os capítulos 7 e 8, os problemas de Regressão.

Considerando o esquema básico de um projeto de Data Science, também já apresentado no capítulo 1, o foco deste e dos próximos capítulos é na etapa de Modelagem e Inferência, ou seja, a escolha do modelo mais adequado para resolver o problema em questão. Resumidamente, esta etapa consiste de 3 tarefas:

1. Elencar os modelos possíveis e passíveis para cada tipo de problema;

2. Estimar os parâmetros que compõem os modelos, baseando-se nas instâncias e variáveis pré-processadas;
3. Avaliar os resultados de cada modelo, usando métricas e um processo justo de comparação

## 5.1 PROBLEMAS DE CLASSIFICAÇÃO

A Classificação é uma das tarefas de Data Science mais importantes e mais populares. Conforme já mencionado, problemas de Classificação utilizam o aprendizado supervisionado: apresentamos, para o algoritmo, um número suficiente de exemplos (também chamados de registros ou instâncias) de entradas e saídas desejadas (já classificadas previamente). O objetivo do algoritmo é aprender uma regra geral que mapeie as entradas nas saídas corretamente. Os dados de entrada podem ser divididos em dois grupos: *X*, com os atributos a serem utilizados na determinação da classe de saída (também chamados de atributos *previsores* ou de *predição*) e *Y*, com o atributo para o qual se deseja fazer a predição do valor da classe (também chamado de *atributo-alvo* ou *target*). Em problemas de Classificação, o atributo-alvo é sempre categórico.

Podemos definir um problema de Classificação informalmente como a busca por uma função matemática que permita associar corretamente cada exemplo $X_i$ de um conjunto de dados a um único rótulo categórico, $Y_i$, denominado *classe*. Uma vez identificada, esta função pode ser aplicada a novos exemplos para prever as classes em que eles se enquadram. A figura a seguir ilustra este problema.

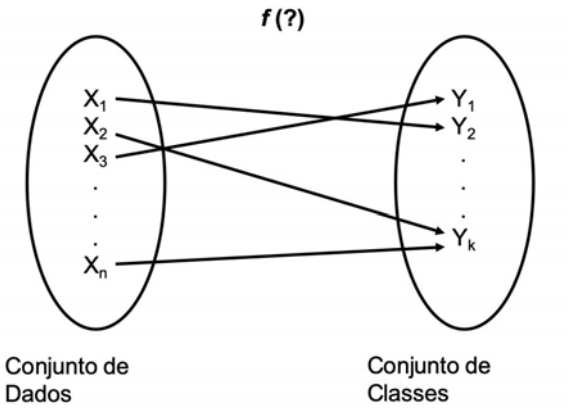



Figura 5.1: Problema de Classificação

Formalmente, podemos definir um problema de Classificação como: dada uma coleção de exemplos *f*, obter uma função (hipótese) *h* que seja uma aproximação de *f*. A imagem de *f* é formada por rótulos de classes retirados de um conjunto finito e toda hipótese *h* é chamada de **classificador**. O aprendizado consiste, então, na busca da hipótese *h* que mais se aproxime da função original *f*.

O fluxo de um problema de Classificação pode ser representado da seguinte forma: a partir de uma base de dados rotulada (para cada exemplo, conhecemos a sua respectiva classe), geram-se dois subconjuntos disjuntos: a base de treino (contendo, por exemplo, 70% dos dados originais) e a base de teste (contendo, por exemplo, 30% dos dados originais), de acordo com as estratégias de amostragens de dados discutidas no capítulo 4.

A base de treino é submetida ao classificador para treinamento do modelo, que é calibrado de acordo com os dados apresentados. Após esta etapa, apresentam-se os exemplos da base de teste para o

modelo, que deverá realizar a predição de suas classes. Comparando-se as classes preditas com as classes verdadeiras da base de teste, pode-se medir a qualidade do modelo, isto é, sua habilidade em classificar corretamente exemplos não vistos durante o treinamento. Este fluxo pode ser resumido pela figura a seguir.

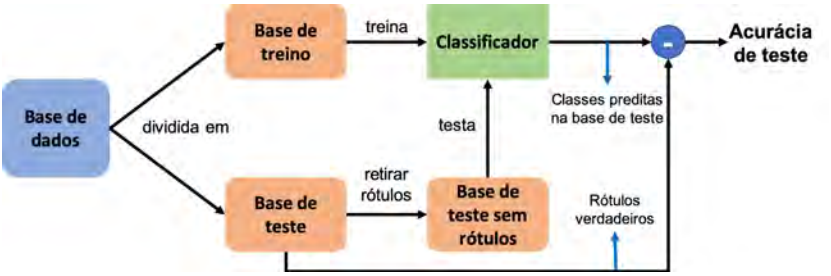



Figura 5.2: Resumo de um problema de Classificação

Existem diversas medidas para estimar o desempenho de um classificador. A mais utilizada é a **acurácia**, ou taxa de acerto do classificador, dada por:

$$Acc(h) = 1 - Err(h)$$

A acurácia é uma função da taxa de erro (ou taxa de classificação incorreta), dada por:

$$Err(h) = \frac{1}{n}\sum_{i=1}^{n} \|y_i \neq h(i)\|$$

Onde:

- O operador $||E||$ retorna 1 se a expressão $E$ for verdadeira e 0 em caso contrário.
- $n$ é o número de exemplos (registros da base de dados)
- $y_i$ é a classe real associada ao i-ésimo exemplo
- $h(i)$ é a classe indicada pelo classificador para o i-ésimo exemplo

Vale a pena observar que, uma vez identificada uma hipótese (uma vez treinado um classificador), esta pode ser muito específica para o conjunto de treino utilizado. Caso este conjunto não corresponda a uma amostra suficientemente representativa da população, este classificador pode ter um bom desempenho no conjunto de treino, mas não no conjunto de teste. Dizemos que ocorreu *overfitting* quando o classificador se ajustou em excesso ao conjunto de treinamento.

Geralmente, o erro de teste é maior que o erro de treinamento. Idealmente, estes erros são próximos. O modelo tem boa capacidade de generalização, sendo capaz de aprender bem exemplos ainda não vistos no treinamento. Se o erro de teste é grande demais em relação ao erro de treino, isso é um indicativo de que pode estar ocorrendo *overfitting* e o modelo está memorizando os padrões de treinamento em vez de descobrir regras ou padrões generalizados.

Em geral, modelos mais simples tendem a generalizar melhor. Um modelo bem ajustado apresentará erros da mesma magnitude tanto nos novos dados quanto nos dados de treino, enquanto um modelo com *overfitting* "memoriza" os dados de treino e apresentará erros muito maiores nos novos dados. Uma típica situação de *overfitting* é ilustrada pela figura a seguir.

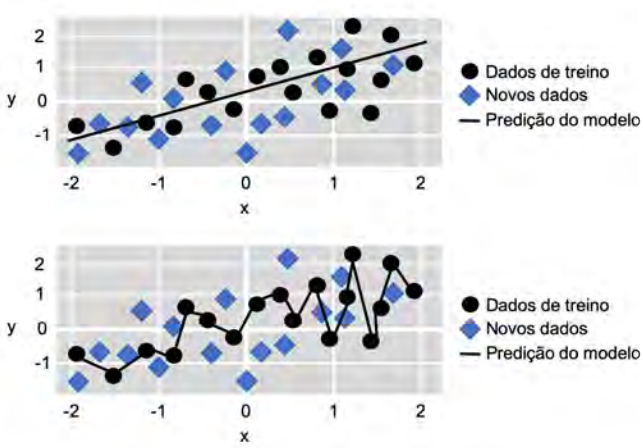



Figura 5.5: Exemplo de overfitting

Outra situação que pode acontecer é se o algoritmo de aprendizado tiver parametrizações inadequadas. Neste caso, diz-se que ocorreu *underfitting*: o classificador se ajustou pouco ao conjunto de treinamento, não sendo adequado para realizar predições no conjunto de teste.

Na aprendizagem supervisionada, ao mesmo tempo em que o modelo preditivo precisa ser suficientemente flexível para aproximar os dados de treinamento, o processo de treinamento deve evitar que o modelo absorva os ruídos da base. O modelo deve capturar as regularidades dos dados de treinamento, mas também generalizar bem para dados desconhecidos (conjunto de teste). A solução para este problema é escolher o momento adequado para interromper o treinamento e/ou utilizar validação cruzada, que será detalhada ainda neste capítulo. A figura a seguir ilustra este momento através da linha pontilhada, imediatamente antes de o erro de teste (erro de generalização) começar a

aumentar.

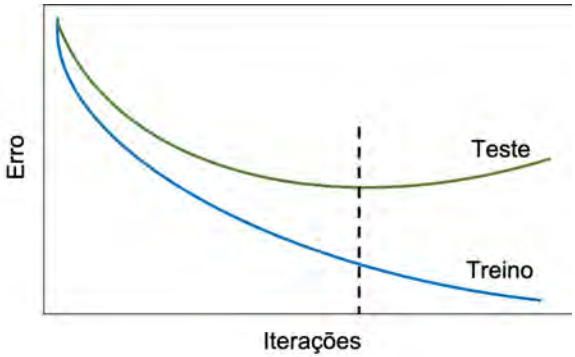



Figura 5.6: Momento adequado para interromper o treinamento

Já a figura a seguir ilustra o comportamento de um modelo possivelmente adequado para o problema, de um modelo com excesso de complexidade para o problema (*overfitting*) e de um modelo com falta de complexidade para o problema (*underfitting*).

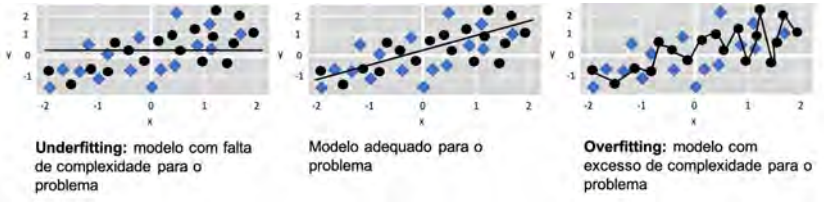

Underfitting: modelo com falta de complexidade para o problema
Modelo adequado para o problema
Overfitting: modelo com excesso de complexidade para o problema

Figura 5.7: Comportamento de modelos

Apesar de a acurácia ser a métrica de desempenho mais utilizada em problemas de Classificação, também é muito comum utilizar a **matriz de confusão** para calcular outras métricas. A matriz de confusão oferece um detalhamento do desempenho do modelo de Classificação, mostrando, para cada classe, o número de classificações corretas em relação ao número de classificações

indicadas pelo modelo.

A partir dela, é possível verificar o número de Falsos Positivos, (FP - também conhecido como Alarme Falso; ou seja, quando o resultado esperado é negativo, mas o modelo resulta em positivo) e Falsos Negativos (FN - também conhecido como Alarme Defeituoso; ou seja, quando o resultado esperado é positivo, mas o modelo resulta em negativo), bem como os números de Verdadeiros Positivos, (VP, quando o resultado esperado é positivo e o modelo resulta em positivo) e Verdadeiros Negativos (VN, quando o resultado esperado é negativo e o modelo resulta em negativo).

As figuras a seguir ilustram, respectivamente, a matriz de confusão para um problema de classificação binária (no qual há somente duas classes possíveis) e para um problema de classificação múltipla (no qual há mais de duas classes possíveis).

| Classes | Predita C1 | Predita C2 |
|---|---|---|
| Verdadeira C1 | Verdadeiros Positivos | Falsos Negativos |
| Verdadeira C2 | Falsos Positivos | Verdadeiros Negativos |

Figura 5.8: Matriz de confusão para problema de classificação binária

| Classes | Predita C1 | Predita C2 | ... | Predita Cn |
|---|---|---|---|---|
| Verdadeira C1 | M(C1, C1) | M(C1, C2) | ... | M(C1, Cn) |
| Verdadeira C2 | M(C2, C1) | M(C2, C2) | ... | M(C2, Cn) |
| ... | ... | ... | ... | ... |
| Verdadeira Cn | M(Cn, C1) | M(Cn, C2) | ... | M(Cn, Cn) |

Figura 5.9: Matriz de confusão para problema de classificação múltipla

A **Curva ROC** (*Receiver Operating Characteristic*) é outra

métrica de avaliação de desempenho de problemas de Classificação muito utilizada, e contrasta os benefícios de uma classificação correta e o custo de uma classificação incorreta. A proporção de exemplos classificados corretamente como positivos é dada pela **Sensibilidade**, ou Taxa de Verdadeiros Positivos (TVP), que representa a capacidade de se identificar corretamente os indivíduos que apresentam a característica de interesse, e é calculada por `VP/(VP+FN)`.

Por sua vez, a **Especificidade** mede a quantidade de negativos verdadeiros corretamente classificados, e representa a capacidade em identificar corretamente os indivíduos que não apresentam a condição de interesse, sendo calculada por `VN/(VN+FP)`.

Na curva ROC, representamos a "Sensibilidade" no eixo y e "1 - Especificidade" (1 menos Especificidade) no eixo x. Quanto mais a curva estiver próxima do canto superior esquerdo, melhor o classificador. A figura a seguir ilustra alguns exemplos de curvas ROC.

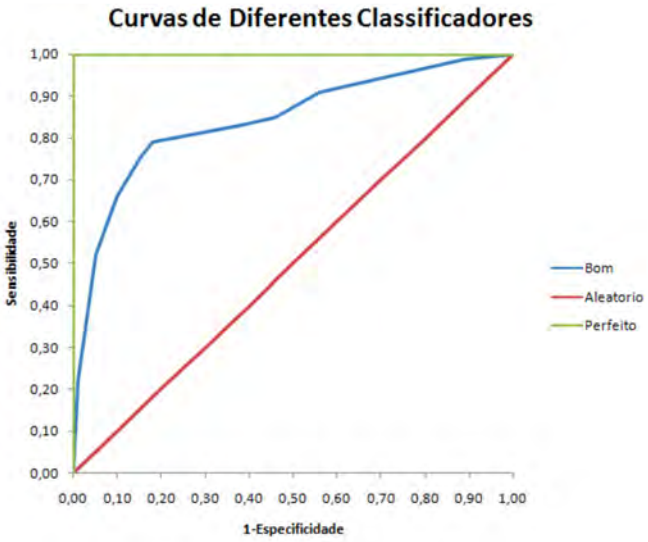



Figura 5.10: Exemplos de curvas ROC. Fonte: (Souza, 2009)

## 5.2 ALGORITMOS

A seguir, serão apresentados alguns algoritmos que podem ser usados para problemas de Classificação.

### Árvore de Decisão

A Árvore de Decisão é um dos modelos preditivos mais simples, e é inspirada na forma como humanos tomam decisão. Uma de suas principais vantagens é apresentar a informação visualmente, de uma forma fácil de entender pelo ser humano. As árvores podem ser usadas para problemas de Classificação - neste caso, chamadas de Árvores de Classificação (apresentadas neste capítulo) - ou Regressão - chamadas Árvores de Regressão (apresentadas no capítulo 7). Basicamente, uma árvore de decisão

usa amostras das características dos dados para criar regras de decisão no formato de árvore, isto é, reduz os dados em um conjunto de regras que podem ser usadas para uma decisão.

As árvores de decisão aliam acurácia e interpretabilidade. Elas possibilitam a seleção automática de variáveis para compor suas estruturas: cada nó interno representa uma decisão sobre um atributo que determina como os dados estão particionados pelos seus nós filhos. Para classificar um novo exemplo, basta testar os valores dos atributos na árvore e percorrê-la até se atingir um nó folha (classe predita). A figura a seguir ilustra uma Árvore de Classificação.

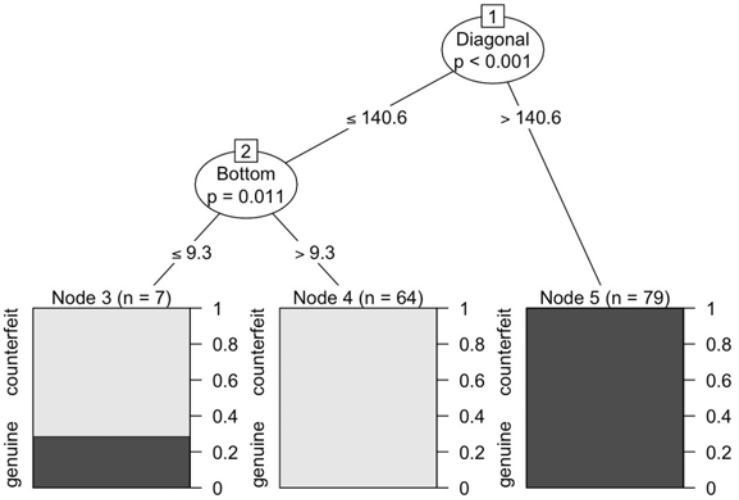



Figura 5.11: Exemplo de Árvore de Classificação

Há diferentes algoritmos para a elaboração de uma Árvore de Classificação. Alguns exemplos são: CHAID, CTree, C4.5, CART e

Hoeffding Tree. Todos os algoritmos são bem parecidos: em geral, a construção da árvore é realizada de acordo com alguma abordagem recursiva de particionamento do conjunto de dados, e a principal distinção está nos processos de seleção de variáveis, critério de particionamento e critério de parada para o crescimento da árvore.

**Árvore C4.5**

Dentre os algoritmos existentes, um dos mais conhecidos é o C4.5, que utiliza conceitos e medidas de Teoria da Informação, uma disciplina que estuda a quantificação, armazenamento e comunicação da informação. O funcionamento do C4.5 pode ser resumido conforme a seguir: inicialmente, a raiz da árvore contém todo o conjunto de dados com exemplos misturados de várias classes. Um predicado (ponto de separação) é escolhido como sendo a condição que melhor separa ou discrimina as classes. Um predicado é um dos atributos previsores do problema e induz uma divisão do conjunto de dados em dois ou mais conjuntos disjuntos, cada um deles associado a um nó filho. Cada novo nó abrange um subconjunto do conjunto de dados original que é recursivamente separado até que o subconjunto associado a cada nó folha consista inteira ou predominantemente de registros de uma mesma classe.

A fase de construção da Árvore de Classificação no algoritmo C4.5 compreende os seguintes passos:

- Uma árvore é gerada pelo particionamento recursivo do conjunto de dados de treinamento.
- O conjunto de treinamento é separado em dois ou mais subconjuntos por meio da definição de predicados sobre os conjuntos de valores de cada atributo previsor.

- O processo de divisão é repetido recursivamente até que todos ou a maioria dos exemplos de cada subconjunto pertençam a uma classe.

Vale a pena observar que os nós folhas da árvore abrangem todo o conjunto de treinamento, e que todos os nós em uma determinada altura devem ser processados antes do processamento do nível subsequente.

São duas as operações principais durante o processo de construção:

1. Avaliação de pontos de separação em potencial para identificação de qual o melhor entre eles;
2. Criação das partições, usando o melhor ponto de separação selecionado, para os casos pertencentes a cada nó.

Uma vez determinado o melhor ponto de separação de cada nó, as partições podem ser criadas pela simples aplicação do critério de partição selecionado.

No C4.5, para a avaliação dos pontos de separação é utilizado o conceito de **entropia**. A entropia de um conjunto de dados é um valor entre 0 e 1 que representa a sua pureza: se em um determinado conjunto todos os registros são da mesma classe, a entropia é 0, e se cada registro é de uma classe diferente, a entropia é 1. A figura a seguir ilustra este conceito:

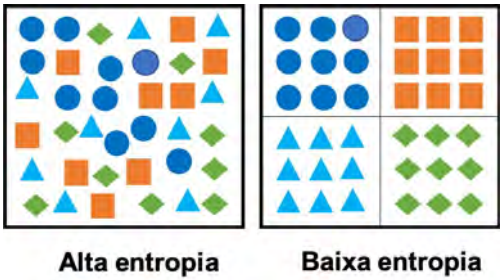



Figura 5.12: Conceito de entropia

O valor da entropia em um nó T é calculado pela fórmula:

$$H(T) = -\sum_{j=1}^{k} \frac{freq(c_j, t)}{|T|} \; x \; log_2 \frac{freq(c_j, t)}{|T|}$$

onde **T** é conjunto de registros de entrada, **freq(cj, t)** é a quantidade de registros da classe $\mathbf{c_j}$ em **T**, **|T|** é o número total de registros em **T** e **k** é o número de classes distintas que ocorrem em **T**.

A seguir, deve ser realizado o cálculo do valor esperado da entropia, representado por $\mathbf{info_{A_i}(T)}$, que consiste no valor esperado da entropia uma vez que *T* ***é dividido com os valores do atributo*** $A_i$***.*** $A_i$ ***deve ser escolhido dentre os atributos ainda não utilizados na definição da árvore. Devemos considerar que existam valores no domínio*** $A_i$ ***e que*** $A_i$ ***induz uma partição sobre*** *T****, resultando em subconjuntos*** $\{T\}^*$.

O valor $\mathbf{info_{A_i}(T)}$ de é calculado pela fórmula:

$$info_{A_i}(T) = j = 1 \sum_{j=1}^{n} \frac{|T_j|}{|T|} \; x \; H(T)$$

onde **T** é o conjunto de registros de entrada, $\mathbf{T_j}$ é cada um dos subconjuntos da partição induzida por $\mathbf{A_i}$ sobre **T**, $\mathbf{|T_j|}$* **é a quantidade de registros em** $T_j$ **e** n** é o número de atributos do conjunto de dados.

Finalmente, realiza-se o cálculo do ganho de informação do atributo $\mathbf{A_j}$ com relação a **T**, que representa a redução na impureza na situação em que os valores de $\mathbf{A_j}$ são usados para subdividir **T**. Após calcular o ganho de informação para cada atributo previsor remanescente, o algoritmo C4.5 seleciona o atributo com maior ganho de informação para associar ao nó da árvore.

A fórmula do ganho de informação (**GInfo($\mathbf{A_j}$,T)**) é dada por:

$$GInfo(A_i, T) = H(T) - \; info_{A_i}(T)$$

As etapas a seguir resumem o algoritmo C4.5:

1. Calcular a entropia do conjunto **T** completo.
2. Para cada atributo:
    - Calcular o ganho de informação.
3. Selecionar o atributo com maior ganho de informação para o nó raiz da árvore.
4. Subdividir o conjunto **T**.
5. Repetir o procedimento para cada nó gerado.

## KNN

O algoritmo KNN (*k-Nearest Neighbours* ou, em português, k-Vizinhos Mais Próximos) é um algoritmo simples de entender e que funciona muito bem na prática tanto para problemas de Classificação quanto para problemas de Regressão. Este é um algoritmo não paramétrico, ele não assume premissas sobre a distribuição dos dados. Sua ideia principal é considerar que os exemplos vizinhos são similares ao exemplo cuja informação se deseja inferir.

A figura a seguir ilustra esta ideia: o KNN considera que os registros do conjunto de dados correspondem a pontos no $\mathbf{R_n}$, em que cada atributo corresponde a uma dimensão deste espaço.

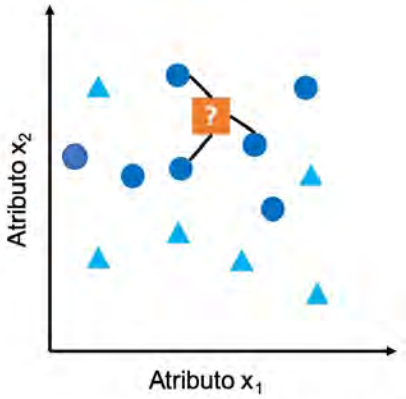



Figura 5.16: Exemplo do funcionamento do KNN

No KNN, inicialmente, o conjunto de dados (rotulado) é armazenado. Quando um novo registro deve ser classificado, ele é comparado a todos os registros do conjunto de treinamento para identificar os **k** (parâmetro de entrada) vizinhos mais próximos

(mais semelhantes) de acordo com alguma métrica de distância. A classe do novo registro é determinada por inspeção das classes desses vizinhos mais próximos, de acordo com a métrica escolhida. Na maioria das implementações do KNN, os atributos são normalizados, para que tenham a mesma contribuição na predição da classe ou do valor.

As etapas a seguir resumem o algoritmo KNN para problemas de Classificação:

1. Definição da métrica de distância, critério de desempate e valor de **k**.
2. Cálculo da distância do novo registro a cada um dos registros existentes no conjunto de referência.
3. Identificação dos **k** registros do conjunto de referência que apresentaram menor distância em relação ao novo registro (mais similares).
4. Apuração da classe mais frequente entre os **k** registros identificados no passo anterior (usando votação majoritária).

Para ilustrar graficamente o funcionamento do algoritmo KNN, considere o problema de determinar o sexo de uma pessoa com base no seu peso e altura. Imagine que já temos 14 exemplos rotulados, nos quais sabemos se cada pessoa pertence ao sexo Masculino ou Feminino. A tabela e figura a seguir ilustram uma amostra dos dados e a distribuição dos indivíduos no espaço.

| id | Peso (kg) | Altura (cm) | Sexo |
|---|---|---|---|
| 1 | 55 | 167 | F |
| 2 | 52 | 160 | F |
| 3 | 48 | 155 | F |
| ... | ... | ... | ... |
| 12 | 70 | 175 | M |
| 13 | 74 | 170 | M |
| 14 | 80 | 180 | M |

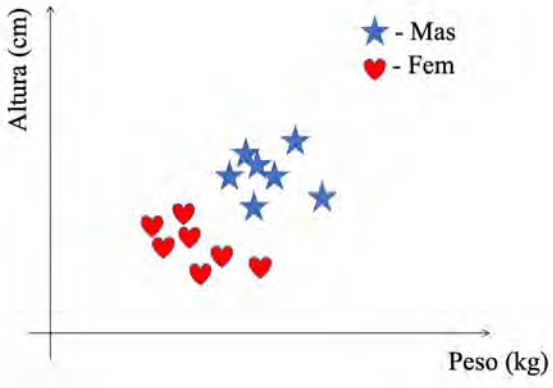



Figura 5.18: Dados do problema

Imagine que neste momento chega um novo indivíduo, com 68 kg e altura 186 cm, e deseja-se determinar o sexo deste indivíduo. A figura a seguir ilustra a localização deste novo indivíduo no espaço.

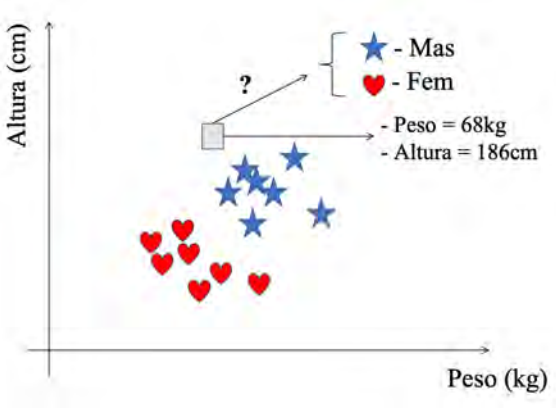



Figura 5.19: Problema de determinar o sexo do novo indivíduo

Podemos então utilizar o KNN para determinar o sexo deste novo indivíduo, que é dado pela informação dos **k** vizinhos mais próximos. Precisamos então determinar o valor de **k**.

Para k = 1:

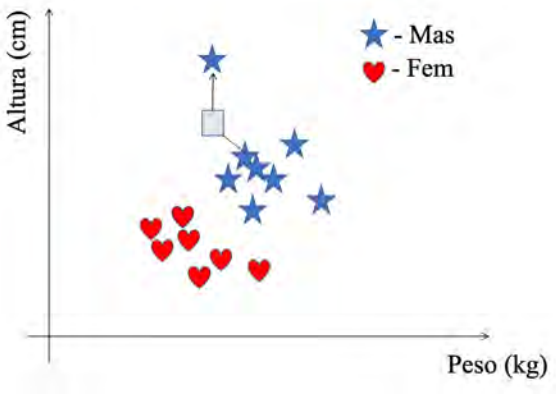



Figura 5.20: Solução com KNN e k = 1

Para k = 3:

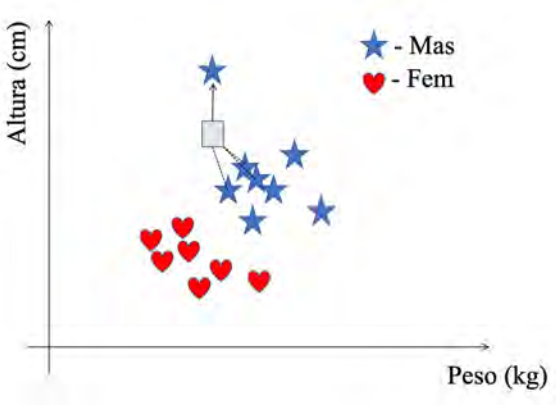



Figura 5.21: Solução com KNN e k = 3

Para k = 5:

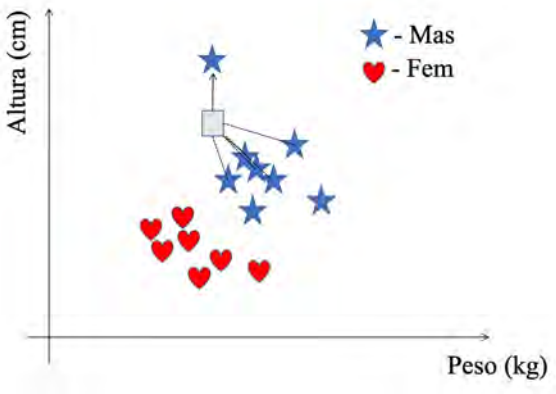



Figura 5.22: Solução com KNN e k = 5

Para os 3 valores de **k** utilizados, o sexo deste indivíduo seria Masculino. Porém, devemos considerar as seguintes perguntas:

1. Que tipo de distância usar?
2. Qual o valor adequado de **k**?

Para a primeira pergunta, diversas medidas de distância podem ser utilizadas, sendo as mais comuns distância Euclidiana, Manhattan e Minkowski. Suas fórmulas estão representadas na figura a seguir:

**Euclidiana:** $d(x_j, y_j) = \sqrt{\sum_{j=1}^{J}(x_j - y_j)^2}$

**Manhattan:** $d(x_j, y_j) = \sum_{j=1}^{J}|x_j - y_j|$

**Minkowski:** $d(x_j, y_j) = \max(|x_j - y_j|)$

Figura 5.23: Distâncias utilizadas

As figuras a seguir ilustram o cálculo de distância usando as três medidas apresentadas:

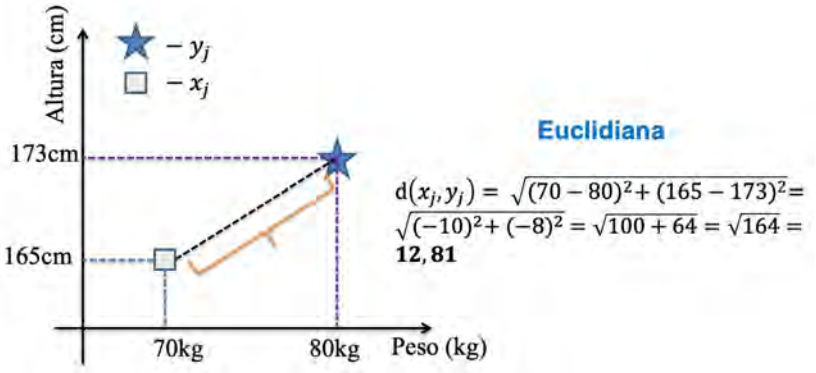



Figura 5.24: Distância Euclidiana

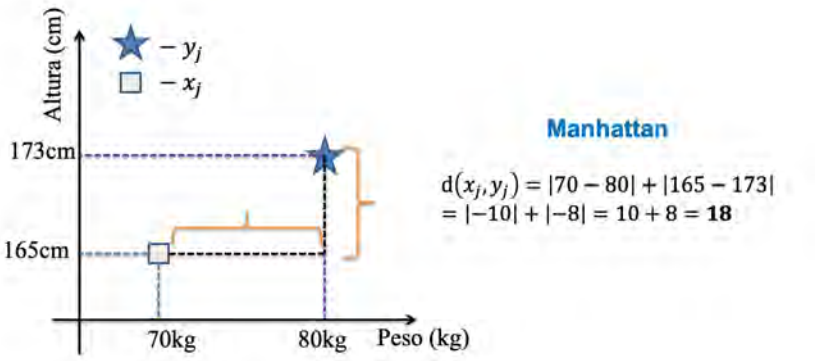



Figura 5.25: Distância de Manhattan

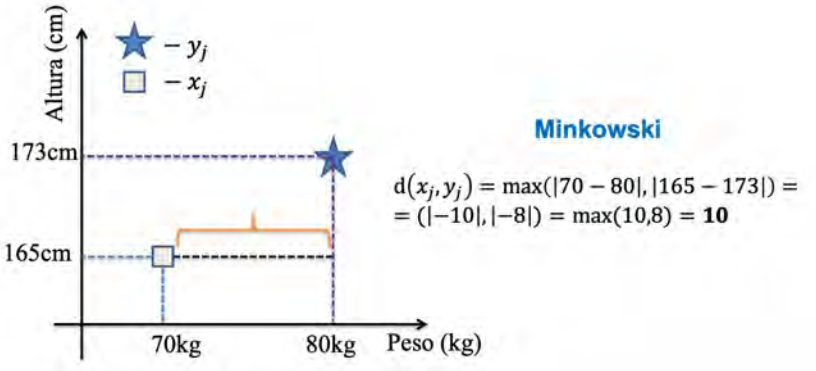



Figura 5.26: Distância de Minkowski

Dependendo da distância escolhida podemos obter um resultado diferente. A decisão de qual distância escolher deve ser experimental: devem ser criados diversos modelos diferentes, variando os parâmetros, e aquele que apresentar melhor resultado (considerando, por exemplo, a acurácia de teste) será o melhor candidato.

Quanto à segunda pergunta (valor adequado de **k**), **k** é

normalmente determinado em função do conjunto de dados. Em geral, quanto maior o valor de **k**, menor o efeito de eventuais ruídos no conjunto de referência, mas valores grandes de **k** tornam mais difusas as fronteiras entre as classes existentes. Geralmente, é uma boa prática a utilização de um valor ímpar, de forma a evitar problema de empate no momento de determinar a classe do novo exemplo. Assim como no caso anterior, recomenda-se que este valor seja decidido experimentalmente. Para tal, uma boa prática é a utilização da validação cruzada.

De forma simplificada, a validação cruzada consiste em selecionar uma fração de dados para treinar o modelo e outra para o seu teste. Por exemplo, pode-se utilizar 80% dos dados para treino e 20% para teste, ou ainda, 70% dos dados para treino e 30% para teste. Imagine que, no exemplo anterior, temos 30 exemplos rotulados disponíveis, e queremos decidir se utilizamos k = 1, k = 3 ou k = 5 para na aplicação do KNN. Se utilizarmos 2/3 dos dados para treino e 1/3 para teste, teremos a seguinte distribuição:

| | id | Peso (kg) | Altura (cm) | Sexo |
|---|---|---|---|---|
| Treinamento | 1 | 55 | 167 | F |
| | 2 | 52 | 160 | F |
| | 3 | 48 | 155 | F |
| | ... | ... | ... | ... |
| | 9 | 58 | 165 | F |
| | 10 | 64 | 169 | F |
| | 11 | 78 | 179 | M |
| | ... | ... | ... | ... |
| | 18 | 70 | 175 | M |
| | 19 | 74 | 170 | M |
| | 20 | 80 | 180 | M |
| Teste | 21 | 57 | 162 | F |
| | 22 | 59 | 161 | F |
| | ... | ... | ... | ... |
| | 29 | 72 | 170 | M |
| | 30 | 69 | 165 | M |

Figura 5.27: Divisão em conjunto de treino e teste

A figura a seguir mostra os dados representados graficamente, destacando os escolhidos para teste:

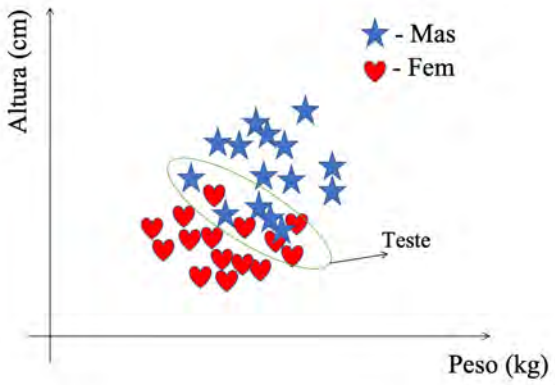



Figura 5.28: Representação gráfica dos dados, destacando o conjunto de teste

A seguir, assume-se que não conhecemos os rótulos dos exemplos do conjunto de teste:

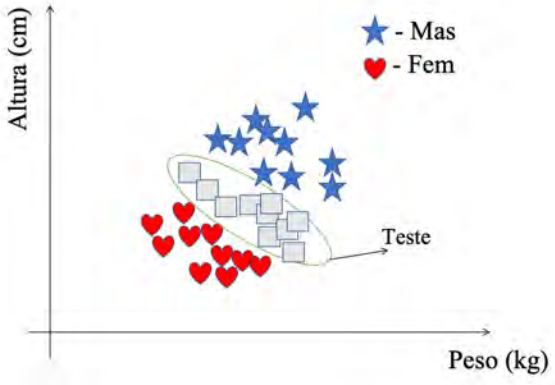



Figura 5.29: Representação gráfica dos dados, destacando o conjunto de teste

Finalmente, aplicamos o modelo e calculamos a acurácia de teste para k = 1, k = 3 e k = 5. Imagine que o resultado da acurácia de teste para cada valor de **k** foi:

| k | Acurácia |
|---|---|
| 1 | 6/10 = 60% |
| 3 | 7/10 = **70%** |
| 5 | 6/10 = 60% |

Assim, seleciona-se o **k** que obtém maior acurácia com os dados de teste (no caso, 3). No caso do KNN, sugere-se variar **k** de 1 a 20 e também as diferentes distâncias disponíveis.

Apesar de ser um algoritmo muito utilizado, o KNN tem algumas limitações: a performance de classificação pode ser lenta em datasets grandes; é sensível a características irrelevantes, uma vez que todas as características contribuem para o cálculo da distância e consequentemente, para a predição; e é necessário testar diferentes valores de **k** e a métrica de distância a utilizar.

## Naïve Bayes (Bayes Ingênuo)

O Naïve Bayes, ou Bayes Ingênuo, é um classificador genérico e de aprendizado dinâmico. É um dos métodos mais utilizados para Classificação, pois é rápido computacionalmente e só necessita de um pequeno número de dados de treinamento. Ele é especialmente adequado quando o problema tem um grande número de atributos (características), e consegue determinar a probabilidade de um exemplo pertencer a uma determinada classe.

As primeiras aplicações comerciais do Naïve Bayes foram na década de 1990, com o objetivo de filtrar spam de e-mails. O método era usado para categorizar textos com base na frequência das palavras encontradas. Atualmente, é um método muito utilizado para previsões em tempo real e/ou em sistemas

embarcados. O Naïve Bayes também é muito utilizado para aplicações de *text mining*, especialmente para classificação de textos e análise de sentimentos nas redes sociais (por exemplo, identificar se o usuário está feliz ou triste ao publicar um texto).

A versão que será apresentada neste livro é uma das mais simples, contudo, é a que se encontra mais implementada nos pacotes do R.

Este método é chamado de ingênuo (naïve, em inglês) porque desconsidera completamente a correlação entre os atributos (características). Por exemplo, em um problema de classificação de animais, se determinado animal é considerado um "Gato" se tiver bigodes, orelhas em pé e aproximadamente 30 cm de altura, o algoritmo não vai levar em consideração a correlação entre esses fatores, tratando cada um de forma independente.

Além disso, o nome do método contém a palavra Bayes porque é baseado no Teorema de Bayes, estando relacionado com o cálculo de probabilidades condicionais. O Teorema de Bayes determina a probabilidade de um evento com base em um conhecimento prévio (a priori) que pode estar relacionado a este evento, e recebe este nome em homenagem ao matemático inglês Thomas Bayes, um dos primeiros estudiosos de probabilidade e teoria de decisão, durante o século XVIII.

Seja X uma observação (ou evidência) proveniente de um conjunto de n atributos. Seja $H$ a hipótese que a observação $X$ pertença a uma determinada classe $C$. Em problemas de Classificação, queremos determinar $P(H|X)$, ou seja, a probabilidade que a hipótese $H$ se aplique a $X$. Em outras palavras, buscamos a probabilidade que a observação $X$ pertença a classe $C$,

dado que conhecemos a descrição de *X*. Assim:

- *P(H|X)* é denominada a probabilidade *a posteriori* de *H* condicionada a *X*. Por exemplo, suponha que tenhamos os dados de idade e renda mensal de clientes, *X* é um cliente de 30 anos com renda mensal R$ 5.000,00 e *H* é a hipótese de este cliente comprar um celular. *P(H|X)* é então a probabilidade de este cliente *X* comprar um celular dado que conhecemos sua idade e renda mensal.
- *P(H)*, por sua vez, é a probabilidade a priori de *H*. É a probabilidade de qualquer cliente, independente de idade, renda mensal ou qualquer outra informação, comprar um celular. Esta probabilidade é baseada em mais informação do que *P(H)*, que é independente de *X*.
- De forma análoga, *P(X|H)* é a probabilidade a posteriori de *X* condicionada a *H*. Por exemplo, a probabilidade de um cliente *X* ter 30 anos e ganhar R$ 5.000,00 dado que este cliente comprou um celular.
- *P(X)*, finalmente, é a probabilidade a priori de *X*, a probabilidade que um cliente do nosso conjunto de dados tenha 30 anos e ganhe R$ 5.000,00.

*P(H)*, *P(X|H)* e *P(X)* podem ser estimadas a partir do próprio conjunto de dados. O Teorema de Bayes é útil para calcular a probabilidade a posteriori *P(H|X)* a partir de *P(H)*, *P(X|H)* e *P(X)*:

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)}$$

Para ilustrar como o Naïve Bayes funciona, vamos utilizar um

exemplo que consiste em definir a liberação de crediário no varejo. Nossa base de dados histórica é composta de *n* clientes com as informações de renda, idade, estado civil e a classificação se é um bom pagador (BP) ou mau pagador (MP). Queremos estimar se um novo cliente João, com renda de R$ 750, 38 anos e casado, será um bom pagador ou mau pagador. A tabela a seguir resume os dados deste problema.

| Cliente | Renda | Idade | Est. Civil | Pagador |
|---|---|---|---|---|
| Ana | 360 | 25 | C | MP |
| Bernardo | 350 | 33 | S | MP |
| Carlos | 1100 | 56 | S | MP |
| ... | ... | ... | ... | ... |
| Xavier | 600 | 33 | S | MP |
| Zé | 800 | 29 | S | BP |
| **João** | **750** | **38** | **C** | **?** |

Figura 5.32: Problema de liberação de crediário no varejo

Primeiramente, vamos calcular a frequência relativa de bons e maus pagadores considerando toda a base de dados. Para tal, calcula-se a frequência relativa, dividindo-se o total de pessoas pertencentes a cada classe pelo total de pessoas na base de dados. O gráfico da figura a seguir ilustra esta informação.

| Cliente | Pagador |
|---|---|
| Ana | MP |
| Bernardo | MP |
| Carlos | MP |
| ... | ... |
| Xavier | MP |
| Zé | BP |
| **João** | **?** |

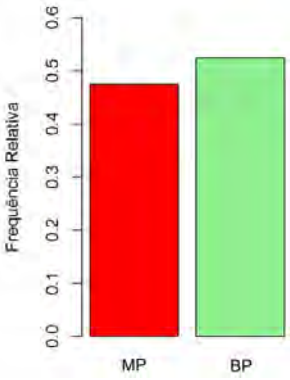



Figura 5.33: Frequência relativa de bons e maus pagadores

A seguir, vamos examinar a idade dos clientes, construindo um histograma de frequência relativa. O gráfico da figura a seguir ilustra esta informação.

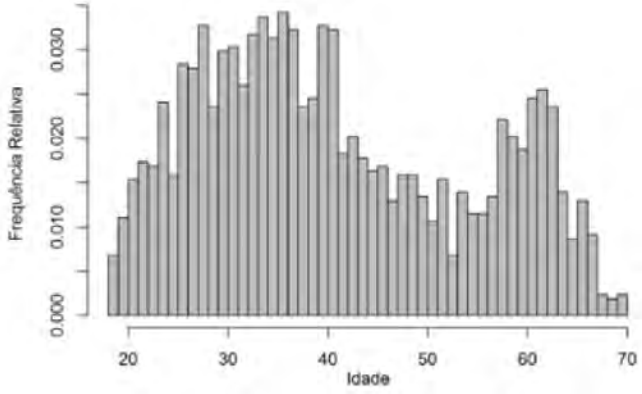



Figura 5.34: Frequência relativa por idade

Para facilitar nossa análise, vamos verificar a idade dos clientes por classe, como mostra a figura a seguir. Em verde, estão representados os bons pagadores e em vermelho os maus

pagadores. As regiões em marrom representam a interseção das duas classes.

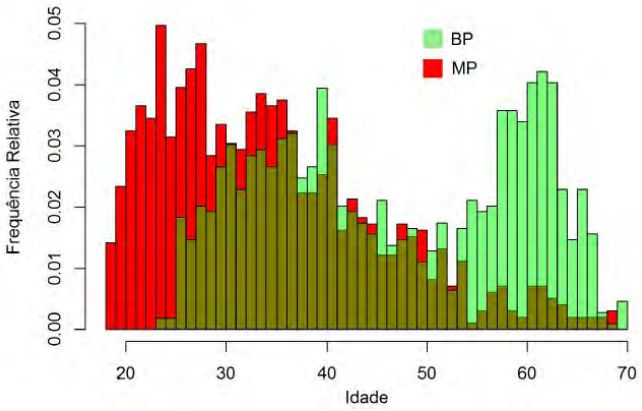



Figura 5.35: Frequência relativa por idade, segmentada por classe

Recapitulando nosso problema, queremos saber qual a probabilidade do cliente João com a idade de 38 anos pertencer às classes MP ou BP. O Naïve Bayes responde este problema calculando a probabilidade de João pertencer a uma das classes em função da frequência da sua idade (38 anos) nos grupos BP e MP e do histórico de bons e maus pagadores. A figura a seguir ilustra o problema.

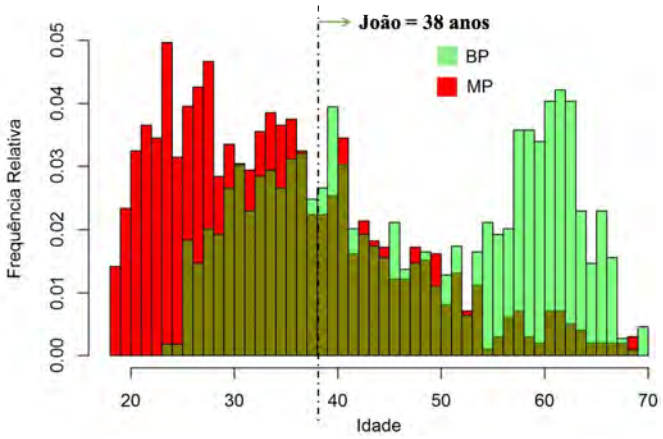



Figura 5.36: Qual a probabilidade de João, com 38 anos, pertencer às classes BP e MP?

Assim, surgem duas novas perguntas:

- Como estimar a frequência histórica de bons e maus pagadores e a frequência da idade 38 anos nos dois grupos?
- Como usar ambas as informações para inferir a probabilidade de João pertencer ao grupo BP e MP?

Vamos tratar primeiramente a primeira parte pergunta 1. Já sabemos que a frequência histórica de bons e maus pagadores pode ser estimada dividindo o total de pessoas que pertençam a esta classe pelo total de pessoas da nossa base de dados histórica. Assim:

$$f(BP) = \frac{\#clientesBP}{\#\ clientes} = 0{,}53$$

$$f(MP) = \frac{\#clientesMP}{\#\ clientes} = 0{,}47$$

Para tratar a segunda parte da pergunta 1, que é estimar a

frequência da idade 38 anos nos dois grupos, apresentaremos dois métodos possíveis. O primeiro método, por histograma, verifica a frequência relativa correspondente a 38 anos nos histogramas individuais de bons e maus pagadores, como ilustram as figuras a seguir.

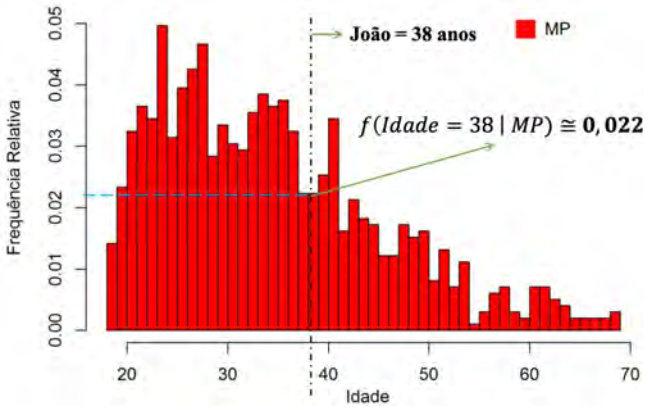



Figura 5.38: Frequência relativa correspondente a 38 anos para maus pagadores usando histograma

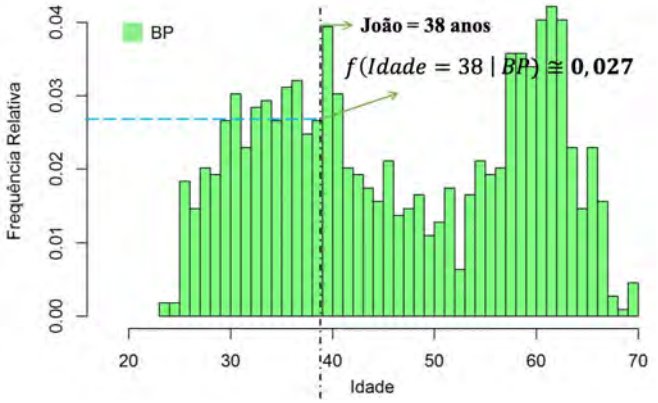



Figura 5.39: Frequência relativa correspondente a 38 anos para bons pagadores usando histograma

Assim, temos que:

$$f(idade = 38|MP) = 0{,}022$$
$$f(idade = 38|BP) = 0{,}027$$

O segundo método, mais utilizado, consiste em utilizar a função de densidade para estimar a frequência de idade em cada grupo. Este método é mais utilizado na prática e será detalhado mais à frente. As figuras a seguir o resumem:

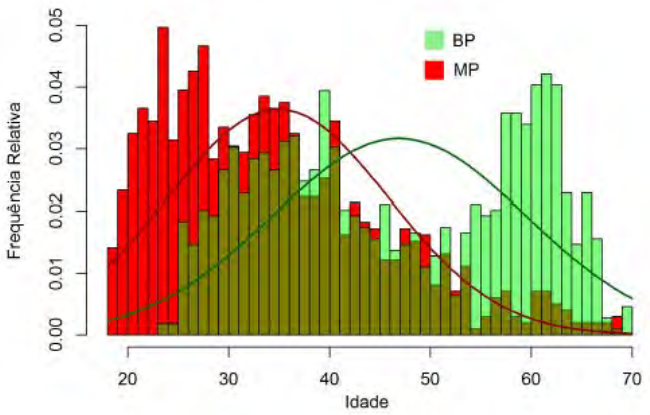



Figura 5.41: Frequência relativa correspondente a 38 anos para bons e maus pagadores usando densidade – passo 1

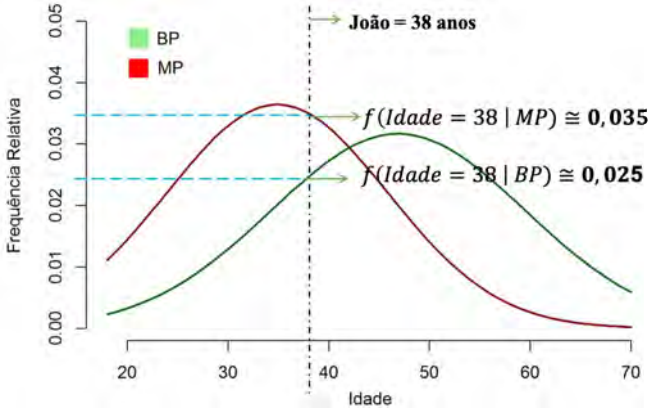



Figura 5.42: Frequência relativa correspondente a 38 anos para bons e maus pagadores usando densidade – passo 2

Utilizando o segundo método, temos que:

$$f(idade = 38|MP) = 0,035$$
$$f(idade = 38|BP) = 0,025$$

Resumindo, até o momento temos estimadores para as frequências de bons/maus pagadores e de uma idade no grupo de bons/maus pagadores:

$$f(MP) = 0,47$$
$$f(BP) = 0,53$$
$$f(idade = 38|MP) \cong 0,035$$
$$f(idade = 38|BP) \cong 0,025$$

Vamos pensar em como podemos aproveitar estas informações para a classificação de João em uma das duas classes. O que

queremos é calcular:

$$f(MP|idade = 38) = ?$$
$$f(BP|idade = 38) = ?$$

Para tal, vamos utilizar a Regra de Bayes, que é baseada no Teorema de Bayes e afirma que a probabilidade *a posteriori* de uma hipótese ser validada pelos dados (*P(H|X)*) é proporcional a distribuição de probabilidade dos dados (função de verossimilhança – *P(X|H)*) multiplicada pela probabilidade *a priori* da hipótese ser verdadeira (*P(H)*). No nosso exemplo:

$$P(MP|idade = 38) \propto P(idade = 38|MP) * P(MP)$$

É importante ressaltar que a probabilidade *a priori* (no exemplo, *P(MP)*) é preconcebida antes dos fatos, o que nesse caso é expresso no desprezo da idade do indivíduo. A verossimilhança representa a compatibilidade entre a evidência (idade = 38 anos) e a classe MP. Por sua vez, a probabilidade *a posteriori* (*PMP|idade=38*) é resultante da combinação entre a observação dos fatos (verossimilhança) e a visão "preconceituosa" sobre as classes.

De forma análoga, obviamente:

$$P(BP|idade = 38) \propto P(idade = 38|BP) * P(BP)$$

Para exemplificar a Regra de Bayes, considere um exemplo em que se deseja prever se o cliente é *BP* ou *MP* dada a sua categoria de idade (Júnior ou Sênior). Temos na base de dados histórica 4

clientes já classificados e queremos classificar o cliente *E*. A tabela a seguir ilustra este problema.

| Id | Fx. Etária | Pagador |
|---|---|---|
| A | Senior | BP |
| B | Junior | MP |
| C | Senior | BP |
| D | Junior | MP |
| E | | ? |

Figura 5.48: Exemplo da Regra de Bayes

Primeiramente, calculamos a probabilidade de *E* se enquadrar em cada uma das duas classes:

$$P(E = MP) \propto \frac{\#MP}{\#clientes} = 2/4$$
$$P(E = BP) \propto \frac{\#BP}{\#clientes} = 2/4$$

Vamos inserir um novo fato, a verossimilhança. Suponhamos que agora sabemos que o cliente *E* é Sênior. Vamos então calcular as probabilidades:

$$P(E = MP | idade = Sênior) \propto ?$$
$$P(E = BP | idade = Sênior) \propto ?$$

Aplicando a Regra de Bayes, sabemos que basta multiplicar a verossimilhança pela probabilidade *a priori* para calcular a probabilidade a posteriori. Assim:

$$P(E = MP|idade = Sênior) \propto \frac{\#Sênior\ e\ MP}{\#MP} * \frac{\#MP}{\#clientes} = \frac{0}{2} * \frac{2}{4} = 0$$
$$P(E = BP|idade = Sênior) \propto \frac{\#Sênior\ e\ BP}{\#BP} * \frac{\#BP}{\#clientes} = \frac{2}{2} * \frac{2}{4} = 1/2$$

Vamos fazer uma normalização das probabilidades, para que as duas somadas resultem em 1. Ao se normalizar, têm-se:

$$P(E = MP|idade = Sênior) \propto \frac{0}{\frac{1}{2} + 0} = 0$$
$$P(E = BP|idade = Sênior) \propto \frac{1/2}{\frac{1}{2} + 0} = 1$$

Já temos a nossa decisão: como `P(E = BP|idade = Senior) > P(E = MP|idade = Senior)` , então, o Cliente *E* tem maiores chances de pertencer ao grupo *BP*.

Vamos então, finalmente, usar a Regra de Bayes para calcular a probabilidade de João, com 38 anos, pertencer às classes *BP* e *MP*:

$$P(MP|idade = 38) \propto P(idade = 38|MP) * P(MP)$$
$$P(MP|idade = 38) \propto 0{,}035 * 0{,}47 = 0{,}016$$

$$P(BP|idade = 38) \propto P(idade = 38|BP) * P(BP)$$
$$P(BP|idade = 38) \propto 0{,}025 * 0{,}53 = 0{,}013$$

Como pendências finais, precisamos ainda responder:

- Como definir uma função densidade de probabilidade para cada grupo?
- Como lidar com mais informação além da idade, por

exemplo, renda, estado civil etc.?

Para definir uma função densidade de probabilidade para cada grupo, vamos primeiro pensar em um grupo por vez, observando o histograma da frequência relativa da idade de cada uma das classes, ilustrados pela figura a seguir.

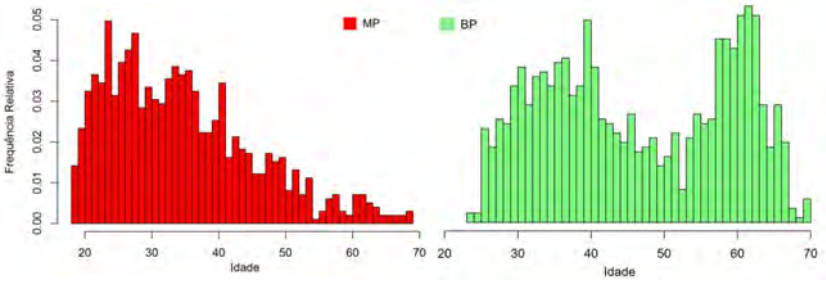



Figura 5.54: Frequência relativa da idade das classes MP e BP

A próxima etapa é escolher um tipo de distribuição que mais se aproxime dos nossos dados. Usualmente, opta-se pela distribuição normal, uma das distribuições de probabilidade mais utilizadas para modelar fenômenos naturais. A distribuição normal tem dois parâmetros: média ($\mu$) e variância ($\sigma^2$). Matematicamente, a função de distribuição normal é expressa por:

$$f_x(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{\sigma^2}}$$

Definido que vamos utilizar a distribuição normal para aproximar nossos dados, primeiramente é necessário calcular a média e a variância da amostra. Em seguida, parametrizamos a função `Normal` com estes valores, obtendo a distribuição normal para a idade e BP. As figuras a seguir ilustram este processo.

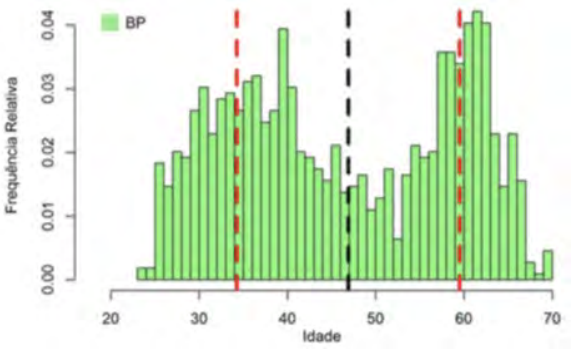



Figura 5.56: Cálculo da média e variância da amostra

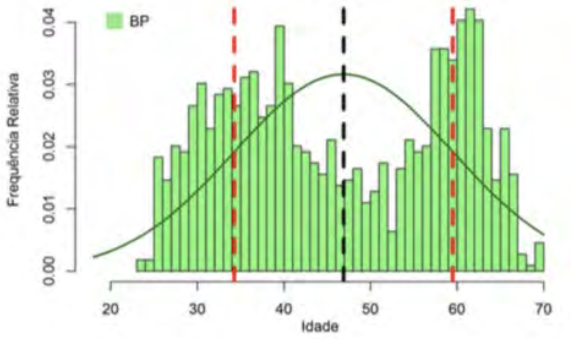



Figura 5.57: Parametrização da função Normal

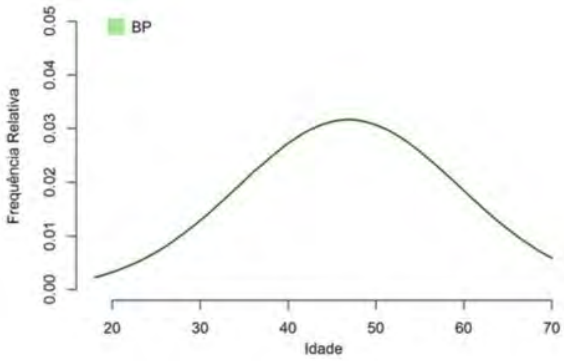



Figura 5.58: Distribuição normal para idade e BP

Agora que já sabemos como definir uma função densidade de probabilidade para cada grupo, precisamos saber como lidar com mais informação além da idade. No problema original, tínhamos também as informações de renda e estado civil. Vale lembrar as informações que já temos:

$$f(MP) = 0{,}047$$
$$f(BP) = 0{,}053$$
$$f(idade = 38|MP) \cong 0{,}035$$
$$f(idade = 38|BP) \cong 0{,}025$$
$$f(MP|idade = 38) = 0{,}016$$
$$f(BP|idade = 38) = 0{,}013$$

E agora, queremos:

$$f(MP|idade = 38\ e\ renda = 750) = ?$$
$$f(BP|idade = 38\ e\ renda = 750) = ?$$

Para calcular as respostas, basta fazer o cálculo de maneira desacoplada:

$$P(MP \mid Idade = 38, Renda = 750)$$
$$\propto f(Idade = 38 \mid MP) * f(Renda = 750 \mid MP) * f(MP)$$

A figura a seguir ilustra este cálculo.

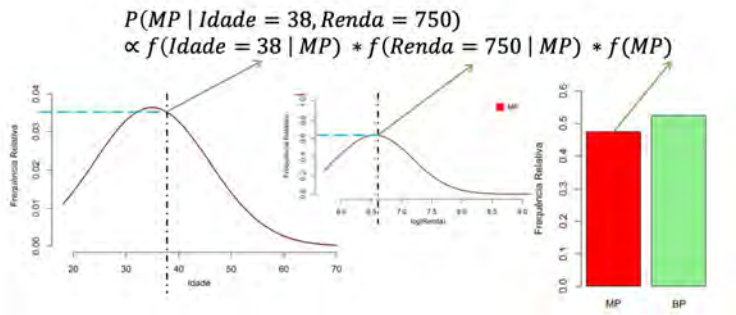



Figura 5.62: Cálculo da probabilidade de o indivíduo João, com idade 38 e renda 750, ser mau pagador

Aplicando o Naïve Bayes, após uma normalização, temos o resultado ilustrado na tabela a seguir.

| Cliente | Renda | Idade | Pagador | P(Cliente=MP) | Decisão |
|---|---|---|---|---|---|
| Ana | 360 | 25 | MP | 0,6 | MP |
| Bernardo | 350 | 33 | MP | 0,3 | BP |
| Carlos | 1100 | 56 | MP | 0,6 | MP |
| ... | ... | ... | ... | ... | ... |
| Xavier | 600 | 33 | MP | 0,6 | MP |
| Zé | 800 | 29 | BP | 0,1 | BP |
| **João** | **750** | **38** | **?** | **0,3** | **BP** |

Figura 5.63: Resultado do problema

Finalmente, apresentamos a definição formal do algoritmo Naïve Bayes. Seja $X(A_1, A_2, \ldots, A_n, C)$ um conjunto de dados. Considere que $c_1, c_2, \ldots, c_k$ são as classes do problema (valores possíveis do atributo alvo $C$) e que $R$ é um registro que deve ser classificado. Sejam ainda $a_1, a_2, \ldots, a_k$ os valores que $R$ assume para os atributos previsores $A_1, A_2, \ldots, A_n$, respectivamente.

O algoritmo consiste em dois passos:

1. Calcular as probabilidades condicionais $P(C=c_i|R)$, $i = 1, 2, \ldots, k$
2. Indicar como saída do algoritmo a classe $c$ tal que $P(C=c|R)$ seja máxima, quando considerados todos os valores possíveis do atributo alvo $C$.

A intuição por trás do algoritmo é dar mais peso para as classes mais frequentes e, conforme já discutimos, é dito ingênuo porque considera como hipótese que os atributos são estatisticamente independentes entre si, o que em muitos casos práticos, não ocorre. Entretanto, na prática o método mostra-se bastante efetivo, mesmo nos casos em que os previsores não são estatisticamente independentes.

## Support Vector Machine (SVM)

*Support Vector Machine* (SVM), ou Máquina de Vetor de Suporte, é um dos algoritmos mais efetivos para classificação. O SVM pode ser aplicado em dados lineares ou não lineares. A primeira pesquisa sobre SVMs foi publicada apenas em 1992, embora os conceitos teóricos relacionados ao tema já vinham sendo explorados desde 1960. Desde a sua criação até os dias de

hoje, é um dos modelos mais populares.

Apesar de o treinamento dos modelos de SVM costumar ser lento, estes modelos exigem poucos ajustes, tendem a apresentar boa acurácia e conseguem modelar fronteiras de decisão complexas e não lineares. Além disso, são menos propensos a *overfitting* se comparados com outros métodos. Como exemplos de aplicações do SVM, podemos citar reconhecimento de manuscritos, reconhecimento de voz, *text mining*, entre outras.

Essencialmente, o SVM realiza um mapeamento não linear para transformar os dados de treino originais em uma dimensão maior. Nesta nova dimensão, o algoritmo busca pelo hiperplano que separa os dados linearmente de forma ótima. Com um mapeamento apropriado para uma dimensão suficientemente alta, dados de duas classes podem ser sempre separados por um hiperplano. O SVM encontra este hiperplano usando vetores de suporte (exemplos essenciais para o treinamento) e margens, definidas pelos vetores de suporte.

Para melhor entendimento do SVM, considere um conjunto de dados de treinamento linearmente separável na forma $\{x_i, y_i\}$, em que $x_i$ corresponde ao vetor de 2 atributos previsores e $y_i \in \{-1,1\}$, as duas classes possíveis do problema. O conjunto de dados de entrada é utilizado para construir uma função de decisão $f(x)$ tal que:

- Se $f(x) > 0$, então $y_i = 1$
- Se $f(x) < 0$, então $y_i = -1$

A figura a seguir ilustra este exemplo.

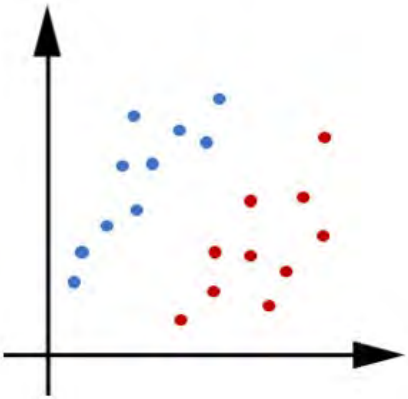


Figura 5.64: Exemplo de problema

Conforme já mencionado, o algoritmo SVM constrói classificadores lineares, que separam os conjuntos de dados por meio de um hiperplano. Considere hiperplano como a generalização do conceito de plano para dimensões maiores que 3. Em um espaço p-dimensional, um hiperplano é um subespaço achatado de dimensão *p-1* que não precisa passar pela origem.

Quando d = 2, o hiperplano é uma reta e sua equação é dada por:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0$$

onde $\beta_1$ e $\beta_2$ são parâmetros da reta que determinam a inclinação da reta e $\beta_0$ o parâmetro que determina o ponto de corte do eixo y. Neste caso, o propósito do SVM é determinar parâmetros da reta, $\beta_0$, $\beta_1$ e $\beta_2$, que permitem separar os conjuntos dos dados de treinamento em duas classes possíveis:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 < 0$$

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 > 0$$

Este exemplo é ilustrado pela figura a seguir.

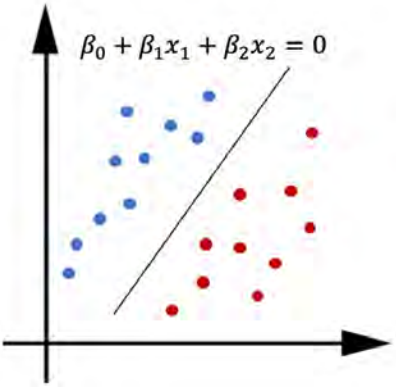



Figura 5.67: Separação dos dados em duas classes através de uma reta

Da mesma forma que um plano tem 2 dimensões e divide um conjunto tridimensional em 2 espaços de dimensão 3, um hiperplano em um espaço n-dimensional tem n-1 dimensões e divide este espaço em 2 subespaços de dimensão n. Assim, para d = p, a equação do hiperplano é:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ \ldots + \beta_n x_n = 0$$

Neste caso, o hiperplano também divide o espaço p-dimensional em 2 metades:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ \ldots + \beta_n x_n < 0$$

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ \ldots + \beta_n x_n > 0$$

Voltando ao exemplo anterior, como afirmamos previamente que o problema é linearmente separável, é possível construir um hiperplano que separe as observações de treino perfeitamente de acordo com seus rótulos de classe. Este hiperplano tem as seguintes propriedades:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 > 0, \; se \; y_i = 1$$

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 < 0, \; se \; y_i = -1$$

O hiperplano pode então ser usado para construir um classificador: o exemplo de teste recebe a classe dependendo de que lado do hiperplano estiver localizado.

É importante notar que, no exemplo anterior, infinitas retas dividem corretamente o conjunto de treinamento em duas classes, conforme ilustra a figura a seguir. O SVM deve, então, realizar um processo de escolha da reta separadora, dentre o conjunto infinito de retas possíveis.

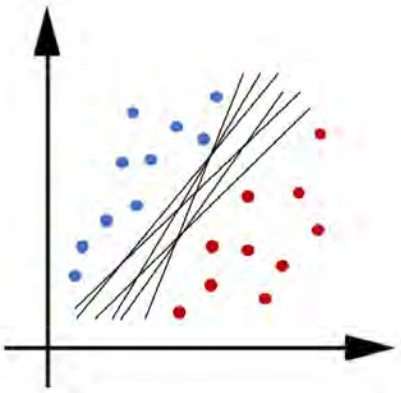


Figura 5.71: Separação dos dados em duas classes através de várias retas

Na figura a seguir, é apresentado apenas um classificador linear (ilustrado pela reta sólida) e duas retas paralelas a este classificador, pontilhadas. Cada uma das retas pontilhadas é movida a partir da posição da reta sólida, e determina quando a reta paralela intercepta o primeiro ponto do conjunto de dados. A margem é a distância construída entre estas duas retas paralelas pontilhadas.

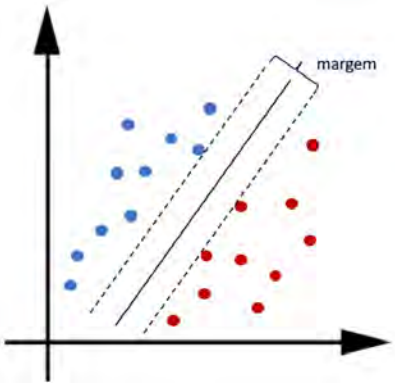



Figura 5.72: Classificador linear e margem

Assim como existem infinitas retas que separam os pontos em duas classes, há diversos tamanhos de margem possíveis dependendo da reta escolhida como classificador. A figura a seguir ilustra dos possíveis classificadores, com dois tamanhos de margem diferentes.

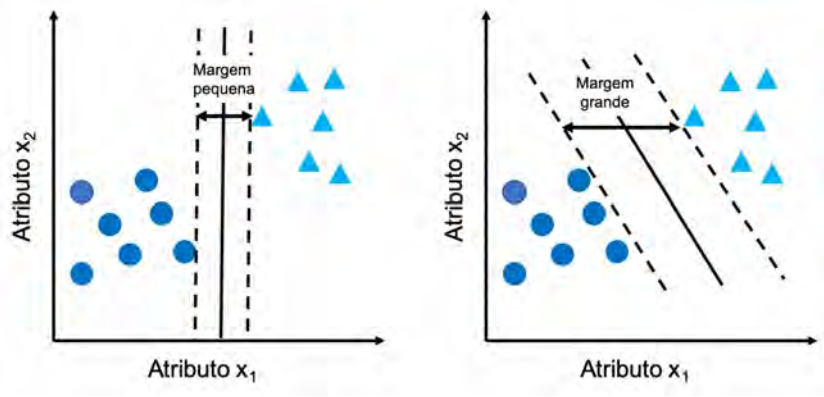



Figura 5.73: Diferentes tamanhos de margem

O classificador associado ao valor máximo de margem é denominado *classificador linear de margem máxima*. Os pontos do conjunto de dados de treinamento que são interceptados pelas linhas da margem são denominados vetores de suporte e são os pontos mais difíceis de classificar. Os vetores de suporte são ilustrados pela figura a seguir. Por construção, todos os vetores de suporte possuem a mesma distância em relação a reta do classificador linear (a metade do comprimento da margem). É importante ressaltar que, apesar de o classificador linear de margem máxima ser geralmente bom, pode haver *overfitting* se o número de dimensões for grande.

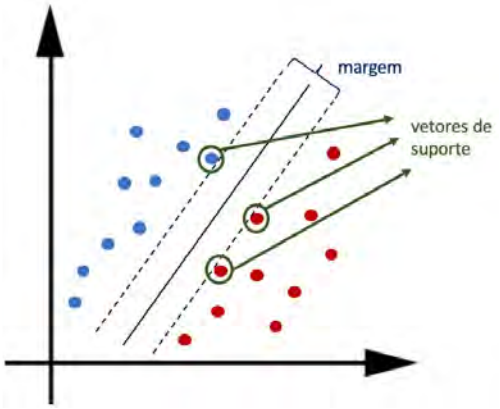



Figura 5.74: Vetores de suporte

O SVM realiza um processo de otimização, por meio do qual são determinados os parâmetros do classificador linear ($\beta_0$, $\beta_1$ e $\beta_2$). O objetivo deste processo de otimização é determinar os valores de $\beta_0$, $\beta_1$ e $\beta_2$ que produzam o valor máximo para o comprimento da margem.

A reta correspondente ao classificador linear é dita ótima porque, se ela for deslocada em alguma das duas direções das retas perpendiculares a ela, a probabilidade é menor de haver um erro de classificação. Assim, a posição do classificador linear correspondente ao comprimento de margem máximo é a mais segura possível com relação a eventuais erros de classificação, e quanto maior a distância de x para o hiperplano, maior a confiança sobre a classe a que x pertence.

A solução do problema de otimização do SVM pode ser obtida usando técnicas de programação quadrática, muito conhecidas na área de Pesquisa Operacional, que consistem em otimizar uma função quadrática sujeita a restrições lineares. A solução deste

problema está fora do escopo deste livro.

Uma vez obtidos os valores dos parâmetros $\beta_0$, $\beta_1$ e $\beta_2$, aplica-se uma função de decisão para classificar um novo exemplo $x_i$. A classe de $x_i$, $y_i$, é dada pelo sinal de *f(x)*:

$$Se\ f(x_i) > 0, então\ y_i = 1$$
$$Se\ f(x_i) < 0, então\ y_i = -1$$

Na prática, entretanto, os dados reais não costumam ser perfeitamente separáveis por um hiperplano. Além disso, o Hiperplano Margem Máxima é extremamente sensível a mudanças em uma única observação, o que sugere que pode ocorrer *overfitting* nos dados de treino. Por isso, podemos querer considerar um classificador baseado em um hiperplano que não separe perfeitamente as duas classes, com o objetivo de aumentar a robustez nas observações individuais e melhorar a classificação na maioria das observações de treino. Ou seja, pode valer a pena classificar erroneamente algumas observações de treino a fim de melhorar a classificação nas observações restantes.

Para tal, usamos o classificador *soft-margin*, que permite que algumas observações do conjunto de treino violem a linha de separação. Neste caso, o hiperplano é escolhido para separar corretamente a maior parte das observações em duas classes, mas pode classificar incorretamente algumas observações. Um conjunto adicional de coeficientes é introduzido na otimização para permitir uma "folga" à margem. Em contrapartida, aumentam a complexidade do modelo.

No exemplo ilustrado pela figura a seguir, o classificador

cometeria erros na classificação para os dois pontos destacados, que podem ser considerados ruído. Para tal, o parâmetro de custo *C* define a "rigidez" da margem e controla o *trade-off* entre o tamanho da margem e o erro do classificador. Quanto maior o valor de *C*, mais pontos podem ficar dentro da margem e maior o erro de classificação, mas menor é a chance de *overfitting*. Se $C = 0$, a margem é rígida e temos o Classificador de Margem Máxima. Na prática, este classificador não produz bons resultados.

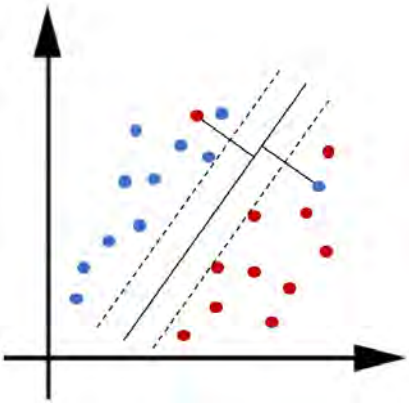


Figura 5.76: Tratamento de ruído

O número total de vetores de suporte depende da quantidade de folga permitida nas margens e da distribuição dos dados. Quanto maior a folga permitida, maior o número de vetores de suporte e mais lenta será a classificação dos dados de teste, pois a complexidade computacional do SVM está relacionada com o número de vetores de suporte.

Na prática, o SVM é implementado usando funções kernel, que, de forma simplificada, são objetos matemáticos que permitem que trabalhemos um espaço de dimensão maior. Neste caso, em

vez de se utilizar as observações em si, é utilizado o seu produto interno. A previsão de um novo exemplo é feita calculando o produto escalar entre o exemplo ($x$) e cada vetor de suporte ($x_i$). Os tipos de kernel mais utilizados são linear, polinomial e radial.

Para um conjunto de dados que não é linearmente separável, o SVM utiliza funções kernel para mapear o conjunto de dados para um espaço de dimensão maior que a original. O classificador é então ajustado neste novo espaço. O SVM é, na verdade, a combinação do classificador linear com um kernel não linear. Este processo é ilustrado pela figura a seguir.

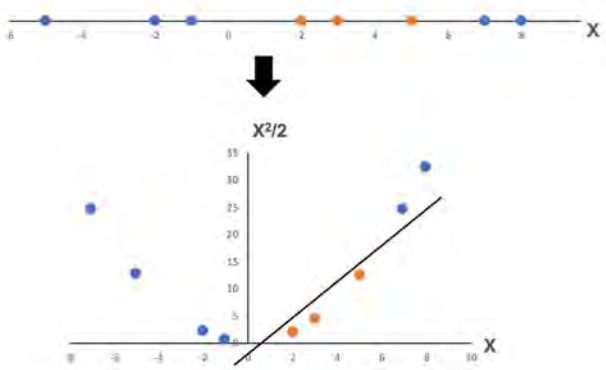



Figura 5.77: Mapeamento de um conjunto não linearmente separável em um linearmente separável

O SVM também pode ser aplicado a problemas de Classificação que envolvem múltiplas classes. Neste caso, usa-se o algoritmo para treinar um modelo de Classificação que informe se o registro é da classe $c_i$ (região positiva) ou se é de alguma outra classe diferente de $c_i$ (região negativa).

Desta forma, constrói-se *p-1* modelos de Classificação, onde *p* é

o número de classes possíveis no problema. Para classificar um novo exemplo, pode-se submetê-lo a cada um dos modelos de classificação gerados, e a classe selecionada para o exemplo é a mais frequentemente atribuída.

*Exemplo*

Para ilustrar o funcionamento do SVM, considere uma base de dados com 2 atributos numéricos $x_1$ e $x_2$ e 1 classe que pode assumir 2 possíveis valores, 1 ou -1. Esta base de dados está representada na tabela a seguir.

| X1 | X2 | Y |
|---|---|---|
| 2,327868056 | 2,458016525 | -1 |
| 3,032830419 | 3,170770366 | -1 |
| 4,485465382 | 3,696728111 | -1 |
| 3,684815246 | 3,846849730 | -1 |
| 2,283558563 | 1,853215997 | -1 |
| 7,807521179 | 3,290132136 | 1 |
| 6,132998136 | 2,140563087 | 1 |
| 7,514829366 | 2,107056961 | 1 |
| 5,502385039 | 1,404002608 | 1 |
| 7,432932365 | 4,236232628 | 1 |

Figura 5.78: Exemplo de base de dados

Observa-se que esta base de dados é linearmente separável. É possível desenhar uma linha reta para separar as classes. Esta situação é ilustrada pela figura a seguir.

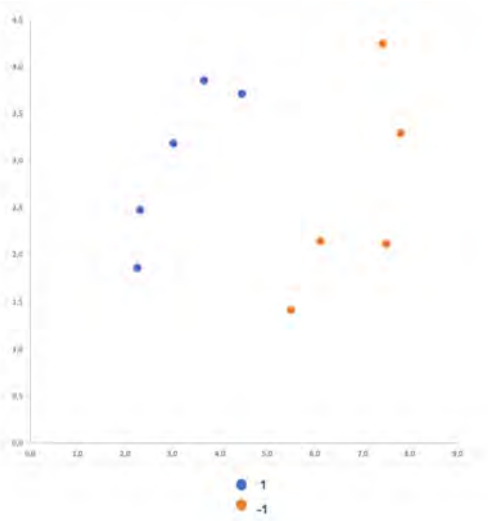



Figura 5.79: Exemplo gráfico da base de dados

Para determinar o classificador linear que melhor separa os exemplos, o SVM resolve um problema de otimização para calcular os coeficientes da reta $w$ e $b$. Conforme já explicado, esta reta pode ser calculada por:

$$w^T x + b = 0$$

onde $w^T$ é o vetor de coeficientes que determinam a inclinação da reta e $x$ é o vetor de atributos da base de dados. Lembrando que no caso de vetores de tamanho 2, o produto escalar $w^T x$ é calculado por:

$$w^T x = w_1 \times x_1 + w_2 \times x_2$$

Assumindo que $b = 0$ para simplificar o exemplo, devemos calcular os coeficientes $w_1$ e $w_2$. Após o fim do processo de otimização, que está fora do escopo deste livro, os coeficientes encontrados são:

$$w_1 = 0{,}552391764$$

$$w_2 = -0{,}724533592$$

e a reta separadora é dada por:

$$(0{,}552391764 \times x_1) + (-0{,}724533592 \times x_2) + 0 = 0$$

Agora que já temos coeficientes para a reta, é possível classificar um novo exemplo. Para tal, pode-se aplicar a seguinte regra:

$$saída = w_1 \times x_1 + w_2 \times x_2$$

$$y = -1, se\ saída < 0$$

$$y = +1, se\ saída \geq 0$$

Suponha então que se deseja classificar um novo exemplo, em que $x_1 = 8$ e $x_2 = 3$. Assim:

$$(0{,}552391764 \times 8) + (-0{,}724533592 \times 3) = 4{,}419134112 - 2{,}173600776 = 2{,}24553334$$

logo, a saída é +1.

## Resumo

Este capítulo apresentou os conceitos teóricos de alguns modelos de aprendizagem supervisionada que podem ser usados para resolver problemas de Classificação. O próximo capítulo apresentará a parte prática.

## Referências bibliográficas do capítulo

(Souza, 2009) - http://crsouza.com/2009/07/13/analise-de-poder-discriminativo-atraves-de-curvas-roc/

CAPÍTULO 6

# PRÁTICAS DE CLASSIFICAÇÃO

No capítulo anterior, apresentamos os conceitos teóricos de alguns modelos de aprendizagem supervisionada que podem ser usados para resolver problemas de Classificação. Este capítulo apresenta exemplos práticos destes modelos. O script deste capítulo está disponível em: https://github.com/tatianaesc/introdatascience/.

## 6.1 ÁRVORES DE CLASSIFICAÇÃO

Para exemplificar a utilização das Árvores de Classificação no R, vamos utilizar a base de dados **banknote**, do pacote **mclust**. Este *dataset* contém dados de notas de 1000 francos suíços, verdadeiras e falsas. Os atributos desta base de dados estão medidos em milímetros (mm), e são: `Length` (comprimento da nota), `Left` (largura da borda esquerda), `Right` (largura da borda direita), `Bottom` (largura da margem inferior), `Top` (largura da margem superior), `Diagonal` (comprimento da diagonal). Para cada exemplo, há duas possíveis classes, armazenadas na coluna `Status`: `genuine` (verdadeira) ou `counterfeit` (falsa). Para executar esta parte prática, copie os comandos a seguir no seu R

Script e execute (ou baixe o script diretamente de https://github.com/tatianaesc/introdatascience/).

Primeiramente, vamos instalar o pacote **mclust** e carregar os dados.

```
> # instala o pacote mclust
> install.packages("mclust")
>
> # carrega a base de dados banknote
> data('banknote', package='mclust')
```

Em seguida, vamos dividir os dados em conjuntos de treino e teste. Para tal, vamos selecionar aleatoriamente e sem reposição (para que não se repitam) 150 exemplos para o conjunto de treinamento. As 50 observações restantes serão usadas para o conjunto de teste.

```
> # verifica o número de linhas do dataset
> numLinhas = nrow(banknote)
>
> # sorteia 150 exemplos dentre todos os exemplos do dataset
> base.treino <- sample(1:numLinhas, 150, FALSE)
>
> # exibe as 5 primeiras linhas sorteadas
> head(base.treino)
 [1]   7  54 117  60 156   8
```

Agora que já separamos os nossos dados em bases de treino e teste, podemos construir o modelo de Classificação que quisermos. O R oferece diversos pacotes com implementações de Árvores de Classificação e, nesta prática, vamos utilizar 3 deles. O primeiro modelo de árvore construído será o C5.0, uma extensão do C4.5, disponível no pacote C50. Este algoritmo usa a medida de entropia para fazer as divisões de nós. Vamos então instalar o pacote C50 e, em seguida, construir o modelo usando os dados de treino.

```
> # instala o pacote C50
> install.packages("C50")
>
> # referencia o pacote C50
> library(C50)
>
> # constrói o modelo usando a variável Status como variável de c
lasse
> # o argumento ~. diz que todos os atributos devem ser utilizado

> modeloC50 <- C5.0(Status ~., data = banknote[base.treino,])
>
> # plota o gráfico da árvore construída
> plot(modeloC50)
```

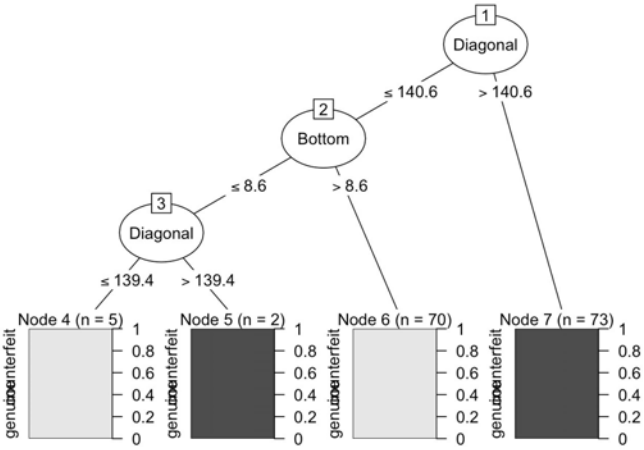



Figura 6.1: Árvore construída

Também podemos construir o mesmo modelo exibindo as regras utilizadas para sua construção no formato `if-then` (ou seja, a representação textual da figura da árvore). Isso facilita a legibilidade no caso de uma árvore grande. Não é possível visualizar o valor da entropia de cada nó, mas esse cálculo é feito internamente pelo algoritmo para definir o melhor particionamento. Para exibir as regras utilizadas, faça:

```
> modeloC50.com.regras <- C5.0(Status ~ ., data = banknote[base.t
reino,], rules=TRUE)
> summary(modeloC50.com.regras)
##
## Call:
## C5.0.formula(formula = Status ~ ., data = banknote[base.treino
, ], rules
##  = TRUE)
##
##
## C5.0 [Release 2.07 GPL Edition]      Wed Jan 17 17:52:26 2018
## -------------------------------
##
## Class specified by attribute `outcome'
##
## Read 150 cases (7 attributes) from undefined.data
##
## Rules:
##
## Rule 1: (70, lift 2.0)
##  Bottom > 8.6
##  Diagonal <= 140.6
##  ->  class counterfeit  [0.986]
##
## Rule 2: (36, lift 1.9)
##  Diagonal <= 139.4
##  ->  class counterfeit  [0.974]
##
## Rule 3: (73, lift 2.0)
##  Diagonal > 140.6
##  ->  class genuine  [0.987]
##
## Rule 4: (54, lift 2.0)
##  Bottom <= 8.6
##  Diagonal > 139.4
##  ->  class genuine  [0.982]
##
## Default class: counterfeit
##
##
## Evaluation on training data (150 cases):
##
##          Rules
##    ----------------
##      No      Errors
```

```
##
##       4    0( 0.0%)   <<
##
##
##     (a)   (b)    <-classified as
##    ----  ----
##      75          (a): class counterfeit
##            75    (b): class genuine
##
##
##  Attribute usage:
##
##  100.00% Diagonal
##   82.67% Bottom
##
##
## Time: 0.0 secs
```

Vamos agora aplicar o modelo construído na base de treinamento, usando a função **predict**, que recebe como argumentos o modelo construído, os dados a serem aplicados e o parâmetro `type="class"` , que indica que queremos que sejam retornados os rótulos de classe. Para tal, faça:

```
> pred.treino <- predict(modeloC50, newdata = banknote[base.trein
o,], type="class")
```

Em seguida, vamos avaliar o modelo, exibindo a matriz de confusão e a acurácia de treino. Como os dados avaliados foram os dados utilizados para construir o modelo, espera-se uma acurácia muito alta. Para aplicar o modelo na base de treinamento, faça:

```
> # cria a matriz de confusão de treino
> matriz.conf.treino <- table(banknote$Status[base.treino], pred.
treino, dnn=c("Classe Observada", "Classe Predita"))
> # exibe a matriz
> print(matriz.conf.treino)
##                 Classe Predita
## Classe Observada counterfeit genuine
##      counterfeit          75       0
##      genuine               0      75
```

```
> # calcula a acurácia
> acc.treino <- (matriz.conf.treino[1,1] + matriz.conf.treino[2,2
]) / (matriz.conf.treino[1,1] + matriz.conf.treino[1,2] + matriz.
conf.treino[2,1] + matriz.conf.treino[2,2]) * 100 # calcula a acu
rácia de treino
> # exibe a acurácia
> print(acc.treino)
## [1] 100
```

Agora vamos ao que interessa: aplicar o modelo nos dados de teste. Para isso, basta utilizar os dados que não estão na base de treino, ou seja, `-base.treino` :

```
> pred.teste <- predict(modeloC50, newdata = banknote[-base.trein
o,], type="class")
```

Após a aplicação, vamos finalmente avaliar o resultado, exibindo a matriz de confusão e a acurácia de teste. Para tal, faça:

```
> matriz.conf.teste <- table(banknote$Status[-base.treino], pred.
teste, dnn=c("Classe Observada", "Classe Predita")) # cria a matr
iz de confusão de teste
> print(matriz.conf.teste) # exibe a matriz
##                 Classe Predita
## Classe Observada counterfeit genuine
##      counterfeit          25       0
##      genuine               0      25
```

```
> # calcula a acurácia
> acc.teste <- (matriz.conf.teste[1,1] + matriz.conf.teste[2,2])
/ (matriz.conf.teste[1,1] + matriz.conf.teste[1,2] + matriz.conf.
teste[2,1] + matriz.conf.teste[2,2]) * 100
> # exibe a acurácia
> print(acc.teste)
## [1] 100
```

Repare que, cada vez que este script for executado, os valores das acurácias serão diferentes, uma vez que o particionamento dos conjuntos de treino e de teste é feito de forma aleatória, então variará a cada execução. Para evitar este problema e possibilitar a reprodutibilidade de resultados, ou seja, garantir que sejam

encontrados os mesmos resultados em uma próxima execução deste mesmo código, podemos configurar uma semente, como veremos em exemplos mais para a frente ao longo deste capítulo.

Vamos agora utilizar outro pacote disponível no R - o pacote **tree** - para construir uma outra Árvore de Classificação, utilizando as mesmas bases de treino e teste já criadas anteriormente. O pacote **tree** permite escolher o critério de particionamento utilizado através do argumento `split` , ao contrário do pacote anterior, que utilizava a entropia como critério padrão. Inicialmente, então, vamos instalar e carregar o pacote:

```
> install.packages("tree")
> library(tree)
```

A seguir, construímos o modelo usando a mesma base de treino já criada anteriormente. As opções de critério de particionamento disponíveis neste pacote são **deviance** e **gini**, que são variações (mas muito similares) da medida de entropia. Neste exemplo, vamos utilizar o critério "deviance":

```
> # constrói o modelo
> modeloTree <- tree(Status ~ ., data = banknote[base.treino,], s
plit="deviance")
> # plota o modelo
> plot(modeloTree)
> text(modeloTree)
```

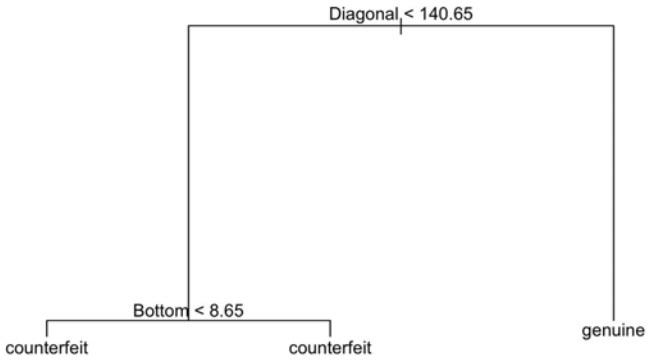



Figura 6.2: Árvore construída

```
> # exibe a performance de classificação
> summary(modeloTree)
##
## Classification tree:
## tree(formula = Status ~ ., data = banknote[base.treino, ], spl
it = "deviance")
## Variables actually used in tree construction:
## [1] "Diagonal" "Bottom"
## Number of terminal nodes:  3
## Residual mean deviance:  0.05698 = 8.376 / 147
## Misclassification error rate: 0.01333 = 2 / 150
```

Vamos aplicar diretamente o modelo nos dados de teste (o que realmente interessa) e exibir a matriz de confusão e a acurácia de teste. Para tal, faça:

```
> # aplica o modelo no conjunto de teste
> pred.teste <- predict(modeloTree, newdata = banknote[-base.trei
no,])
> # guarda as classes preditas
> pred.class <- colnames(pred.teste)[max.col(pred.teste, ties.met
```

```
hod = c("random"))]
> # calcula e exibe a matriz de confusão
> matriz.conf.teste <- table(banknote$Status[-base.treino], pred.
class, dnn = c("Classe Observada", "Classe Predita"))
> print(matriz.conf.teste)
##                 Classe Predita
## Classe Observada counterfeit genuine
##      counterfeit          25       0
##      genuine               0      25

> # calcula e exibe a acurácia
> acc.teste <- (matriz.conf.teste[1,1] + matriz.conf.teste[2,2])
/ (matriz.conf.teste[1,1] + matriz.conf.teste[1,2] + matriz.conf.
teste[2,1] + matriz.conf.teste[2,2]) * 100
> print(acc.teste)
## [1] 100
```

Finalmente, vamos utilizar outro pacote disponível no R para construir uma terceira Árvore de Classificação. Agora vamos usar a função `ctree`, disponível no pacote **party**. Inicialmente, instalamos e carregamos o pacote:

```
> install.packages("party")
> library(party)
```

A seguir, construímos o modelo utilizando a mesma base de treino já criada anteriormente:

```
> modeloCTree <- ctree(Status ~ ., data = banknote[base.treino,])
# constrói o modelo
> plot(modeloCTree)
```

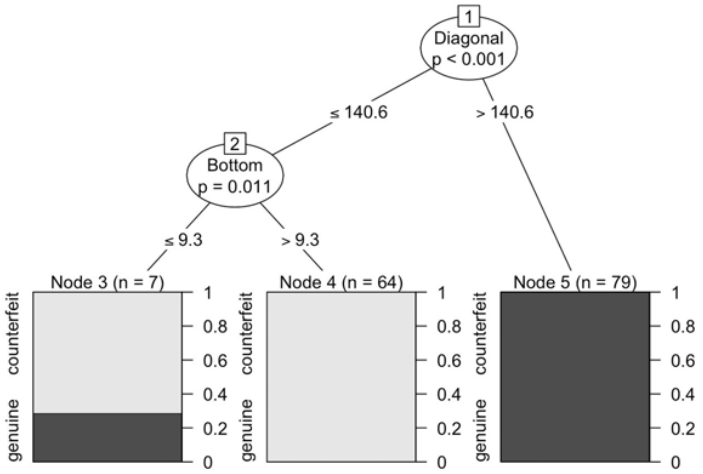



Figura 6.3: Árvore construída

Vamos aplicar diretamente o modelo nos dados de teste e exibir a matriz de confusão e a acurácia de teste:

```
> # aplica o modelo nos dados de teste
> pred.teste <- predict(modeloCTree, newdata = banknote[-base.tre
ino,])
> # calcula e exibe a matriz de confusão
> matriz.conf.teste <- table(banknote$Status[-base.treino], pred.
teste, dnn=c("Classe Observada", "Classe Predita"))
> print(matriz.conf.teste)
##                 Classe Predita
## Classe Observada counterfeit genuine
##      counterfeit          31       0
##      genuine               0      19

> # calcula e exibe a acurácia
> acc.teste <- (matriz.conf.teste[1,1] + matriz.conf.teste[2,2])
/ (matriz.conf.teste[1,1] + matriz.conf.teste[1,2] + matriz.conf.
teste[2,1] + matriz.conf.teste[2,2]) * 100
> print(acc.teste)
## [1] 100
```

Nesta prática, vimos 3 possíveis formas de trabalhar com Árvores de Classificação no R. Na prática, desde que usemos as mesmas bases de treinamento e teste, podemos criar diversos modelos usando os diferentes pacotes e então verificar qual apresenta o melhor resultado, medido pela acurácia de teste. Quando o modelo oferece possibilidade de variação de parâmetros (como o critério de particionamento, disponível no pacote `tree`), podemos comparar o resultado de cada uma destas possíveis variações, para então escolher o modelo mais adequado para o problema. Essa é uma excelente prática e será melhor detalhada no capítulo de conclusão deste livro.

## 6.2 KNN

Para ilustrar o algoritmo KNN em um problema de Classificação, vamos usar o dataset **wine**, disponível no pacote **rebmix**, que contém 178 exemplos com 13 características derivadas da análise química de 3 tipos de vinho provenientes da mesma região da Itália, mas de vinícolas diferentes. 12 dos atributos desta base de dados são contínuos: `Alcohol`, `Malic.Acid`, `Ash`, `Alcalinity.of.Ash`, `Magnesium`, `Total.Phenols`, `Flavanoids`, `Nonflavanoid.Phenols`, `Proanthocyanins`, `Color.Intensity`, `Hue`, `OD280.OD315.of.Diluted.Wines`. 1 atributo é inteiro: `Proline`. A classe é representada pela característica `Cultivar`, que pode ser 1, 2 ou 3, de acordo com a vinícola originária.

Para executar esta parte prática, copie os comandos a seguir no seu R Script e execute (ou baixe o script diretamente de https://github.com/tatianaesc/introdatascience/). Primeiramente, vamos instalar o pacote *rebmix* e carregar os dados:

```
> install.packages("rebmix")
> data("wine", package = "rebmix")
```

Conforme já explicado, para determinar a classe de uma nova instância, o algoritmo KNN calcula a sua distância em relação aos dados com classe já conhecida. Por este motivo, é uma boa prática normalizar a escala das características. Por exemplo, imagine que estamos trabalhando em um problema com características em escalas diferentes: peso (variando de 50 a 150 kg) e nota em uma prova (variando de 0 a 10). Neste caso, suponha dois indivíduos A e B com uma mesma nota e pesos respectivamente de 60kg e 70kg e indivíduos C e D com um mesmo peso e notas respectivamente 6 e 7. A está tão próximo de B quanto C está de D, porém, como a distância entre os pesos de A e B é 10, e entre as notas de C e D é 1, o KNN poderia determinar que C está mais próximo de D do que A de B, uma vez que a distância absoluta é menor. Entretanto, se a característica peso for normalizada em uma escala entre 0 e 10, este problema não ocorrerá, uma vez que as duas características agora estarão na mesma escala e, consequentemente, as distâncias serão as mesmas.

Voltando para o exemplo prático, com a função `boxplot`, é possível verificar que o atributo `Proline` tem claramente um `range` maior que as outras características:

```
> boxplot(wine)
```

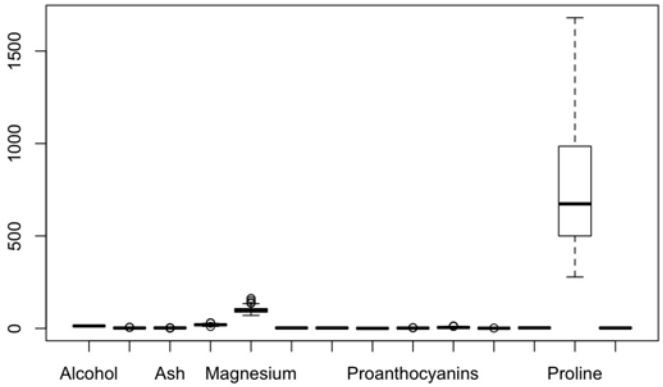



Figura 6.4: Boxplot

Como verificamos que é necessário normalizar os dados, podemos fazer isto com a função `scale` :

```
> # pegamos apenas as características
> atributos <- wine[, 1:13]
> # normalizamos as características usando média 0 e desvio padrã
o 1
> atributos <- scale(atributos)
```

Verificamos no `boxplot` que agora os dados estão normalizados:

```
> boxplot(atributos)
```

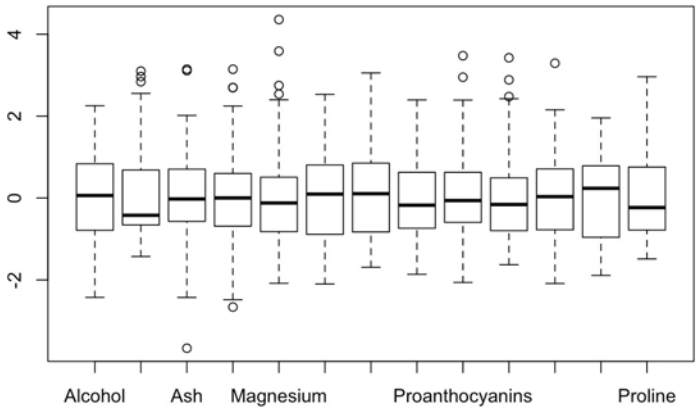



Figura 6.5: Boxplot com dados normalizados

Agora precisamos criar os nossos conjuntos de treino e de teste usando amostragem aleatória sem reposição, assim como fizemos no exemplo da Árvore de Classificação. Vamos então utilizar uma semente, que garantirá que, a cada execução deste script, os mesmos resultados serão gerados mesmo com esta etapa que é aleatória, possibilitando a reprodutibilidade dos resultados. A semente pode ser um valor qualquer, e neste exemplo vamos usar 2016:

```
> set.seed(2016)
```

Para este problema, vamos usar a metade dos dados para treino e a outra metade para teste. Para tal, fazemos:

```
> numLinhas = nrow(atributos)
> baseTreino <- sample(1:numLinhas, 89, replace = FALSE)
```

Uma das formas de estimar um modelo KNN no R é usando a função `knnVCN` , do pacote `knnGarden` . Esta função nos possibilita especificar várias métricas de distância. Vamos então instalar e carregar este pacote:

```
> install.packages("knnGarden")
> library("knnGarden")
```

Vamos construir um primeiro modelo usando K = 2 e a medida de distância de `canberra` , uma medida numérica da distância entre pares de pontos em um espaço vetorial. Com um único comando, conseguimos treinar o modelo com os dados de treino e testá-lo com os dados de teste. A função `knnVCN` receberá como parâmetros, nesta ordem: as características da base de treino, a variável de classe, as características da base de teste, o número de vizinhos e a métrica de distância. Para tal, faça:

```
> modeloKNN <- knnVCN(atributos[baseTreino,],
                      wine$Cultivar[baseTreino],
                      atributos[-baseTreino,],
                      K = 2,
                      method = "canberra")
```

Os valores preditos pelo modelo estarão armazenados na variável `modeloKNN$TstXIBelong` . Vamos mostrá-los em uma matriz de confusão e calcular a acurácia de teste:

```
> matrizConf <- table(modeloKNN$TstXIBelong, wine$Cultivar[-baseT
reino])
> print(matrizConf)
##
##      1  2  3
##   1 27  2  0
##   2  0 34  3
##   3  0  1 22
```

```
> accTeste <- (matrizConf[1,1] + matrizConf[2,2] + matrizConf[3,3
]) / (matrizConf[1,1] + matrizConf[1,2] + matrizConf[1,3] + matri
zConf[2,1] + matrizConf[2,2] + matrizConf[2,3] + matrizConf[3,1]
```

```
+ matrizConf[3,2] + matrizConf[3,3]) * 100
> print(round(accTeste,2))
## [1] 93.26
```

Vamos agora experimentar uma outra medida de distância, a euclidiana, que é a distância simples entre dois pontos, e ver o que acontece com a acurácia de teste. Para tal, basta variar o parâmetro `method`, da função `knnVCN`:

```
> modeloKNN <- knnVCN(atributos[baseTreino,],
                      wine$Cultivar[baseTreino],
                      atributos[-baseTreino,],
                      K = 2,
                      method = "euclidean")
```

```
> matrizConf <- table(modeloKNN$TstXIBelong, wine$Cultivar[-baseT
reino])
> print(matrizConf)
##
##      1  2  3
##   1 27  3  0
##   2  0 34  1
##   3  0  0 24
```

```
> accTeste <- (matrizConf[1,1] + matrizConf[2,2] + matrizConf[3,3
]) / (matrizConf[1,1] + matrizConf[1,2] + matrizConf[1,3] + matri
zConf[2,1] + matrizConf[2,2] + matrizConf[2,3] + matrizConf[3,1]
+ matrizConf[3,2] + matrizConf[3,3]) * 100
> print(round(accTeste,2))
## [1] 95.51
```

É fácil notar que conseguimos melhorar um pouco a acurácia de teste. Agora, vamos continuar usando a distância euclidiana, mas alterar o valor de K para 3 e ver o que acontece com a acurácia de teste. Para tal, basta variar o parâmetro `K`, da função `knnVCN`:

```
> modeloKNN <- knnVCN(atributos[baseTreino,],
                      wine$Cultivar[baseTreino],
                      atributos[-baseTreino,],
                      K = 3,
                      method = "euclidean")
```

```
> matrizConf <- table(modeloKNN$TstXIBelong, wine$Cultivar[-baseT
reino])
> print(matrizConf)
##
##      1  2  3
##   1 27  2  0
##   2  0 35  1
##   3  0  0 24
```

```
> accTeste <- (matrizConf[1,1] + matrizConf[2,2] + matrizConf[3,3
]) / (matrizConf[1,1] + matrizConf[1,2] + > matrizConf[1,3] + mat
rizConf[2,1] + matrizConf[2,2] + matrizConf[2,3] + matrizConf[3,1
] + matrizConf[3,2] + matrizConf[3,3]) * 100
> print(round(accTeste,2))
## [1] 96.63
```

Com esta variação conseguimos aumentar um pouco mais a acurácia de teste. Conforme mencionamos anteriormente, recomenda-se sempre utilizar a validação cruzada para determinar a métrica de distância e o valor de K mais adequados para o seu problema. Esta técnica será detalhada ainda neste capítulo. Para simplificar o exemplo nesta parte do livro, fizemos apenas poucas variações do modelo.

## 6.3 NAÏVE BAYES (BAYES INGÊNUO)

Para ilustrar o algoritmo Naive Bayes em um problema de Classificação, vamos usar agora como dataset uma planilha do Excel em vez de um dataset disponível no R (este arquivo está disponível em https://github.com/tatianaesc/introdatascience). Para executar esta parte prática, copie os comandos a seguir no seu R Script e execute (ou baixe o script diretamente do repositório no GitHub). Primeiramente, vamos importar os dados:

```
> tabela = read.table("BD_aulaclass.csv", sep = ";", dec = ",", h
eader = TRUE)
```

Podemos exibir um sumário dos dados importados, com o comando `summary`. Para tal, faça:

```
> summary(tabela)
##    ID_CLIENTE   ESTC          NDEP            RENDA        TIP
OR
##  Min.   :   1   C: 832   Min.   :0.0000   Min.   : 300.0   NP:
1148
##  1st Qu.: 520   D:  39   1st Qu.:0.0000   1st Qu.: 470.0   P :
 929
##  Median :1039   S:1148   Median :0.0000   Median : 640.0
##  Mean   :1039   V:  58   Mean   :0.1223   Mean   : 969.5
##  3rd Qu.:1558            3rd Qu.:0.0000   3rd Qu.:1150.0
##  Max.   :2077            Max.   :7.0000   Max.   :9675.0
##       VBEM            NPARC            VPARC       TEL
  IDADE
##  Min.   : 300.0   Min.   : 1.000   Min.   : 50.0   N:1811   Mi
n.   :18.0
##  1st Qu.: 404.0   1st Qu.: 6.000   1st Qu.: 66.0   S: 266   1s
t Qu.:31.0
##  Median : 489.0   Median :10.000   Median : 83.0            Me
dian :39.0
##  Mean   : 563.8   Mean   : 8.395   Mean   :102.7            Me
an   :41.2
##  3rd Qu.: 618.0   3rd Qu.:10.000   3rd Qu.:118.0            3r
d Qu.:52.0
##  Max.   :6000.0   Max.   :24.000   Max.   :719.0            Ma
x.   :70.0
##      RESMS          ENTRADA        ADIMPLENTE
##  Min.   :  0.0   Min.   :   0.00   N: 986
##  1st Qu.:  6.0   1st Qu.:   0.00   S:1091
##  Median :  6.0   Median :   0.00
##  Mean   : 30.9   Mean   :  30.66
##  3rd Qu.: 48.0   3rd Qu.:   0.00
##  Max.   :420.0   Max.   :1300.00
```

Vamos armazenar os atributos no objeto `x` e as classes no objeto `y`:

```
> x <- tabela
> x$ADIMPLENTE <- NULL
> y <- tabela$ADIMPLENTE
```

A seguir, vamos sortear 75% dos dados para compor a base de treino, usando uma semente para garantir a reprodutibilidade dos resultados deste script, da mesma forma que fizemos anteriormente:

```
> set.seed(2018)
> N = nrow(x)
> baseTreino <- sample(1:N, N*0.75, FALSE)
```

Vamos então instalar o pacote `e1071`, que contém a função `naiveBayes`, que será utilizada para a construção do modelo para prever `y` (classes) em função de `x` (atributos). Primeiro, instale e referencie o pacote:

```
> install.packages("e1071")
> library("e1071")
```

Em seguida, crie e exiba o modelo:

```
> modeloNB <- naiveBayes(y[baseTreino]~., data = x[baseTreino,])
> show(modeloNB)
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##         N         S
## 0.4662813 0.5337187
##
## Conditional probabilities:
##    ID_CLIENTE
## Y       [,1]     [,2]
##   N 1010.253 609.5600
##   S 1055.004 599.6231
```

```
##
##    ESTC
## Y            C          D          S          V
##   N 0.47520661 0.01515152 0.48071625 0.02892562
##   S 0.30565584 0.02166065 0.64500602 0.02767750
##
##    NDEP
## Y         [,1]      [,2]
##   N 0.07162534 0.3935260
##   S 0.16606498 0.6447966
##
##    RENDA
## Y        [,1]     [,2]
##   N  865.9394 693.2375
##   S 1048.5800 963.9865
##
##    TIPOR
## Y          NP         P
##   N 0.8085399 0.1914601
##   S 0.3261131 0.6738869
##
##    VBEM
## Y       [,1]     [,2]
##   N 581.7755 367.0123
##   S 561.1576 259.8222
##
##    NPARC
## Y       [,1]     [,2]
##   N 9.130854 3.624546
##   S 7.761733 3.403192
##
##    VPARC
## Y       [,1]     [,2]
##   N 101.8857 68.61285
##   S 104.4753 63.41937
##
##    TEL
## Y            N          S
##   N 0.82369146 0.17630854
##   S 0.91215403 0.08784597
##
##    IDADE
## Y       [,1]     [,2]
##   N 35.00138 10.94494
##   S 46.95788 12.65661
```

```
##
##    RESMS
## Y       [,1]     [,2]
##   N 41.35537 61.80022
##   S 22.38147 41.63652
##
##    ENTRADA
## Y       [,1]     [,2]
##   N 16.08815  68.0600
##   S 44.06017 114.9206
```

Vamos finalmente aplicar o modelo na base de teste usando a função `predict`. Para verificar as probabilidades de pertencer a cada classe, usamos o parâmetro `type = "raw"`:

```
> probsTeste <- predict(modeloNB, x[-baseTreino,], type = "raw")
> head(round(probsTeste,3),4)
##          N     S
## [1,] 0.972 0.028
## [2,] 0.429 0.571
## [3,] 0.226 0.774
## [4,] 0.913 0.087
```

Já para verificar as classes preditas, usamos o parâmetro `type = "class"`:

```
> classesTeste <- predict(modeloNB, x[-baseTreino,], type = "clas
s")
> head(classesTeste)
## [1] N S S N S S
## Levels: N S
```

Vamos fazer a avaliação deste modelo. Para tal, vamos utilizar o pacote **caret**, que contém algumas funções que simplificarão bastante esta tarefa. Primeiro, instale este pacote:

```
> install.packages("caret")
> library("caret")
```

Vamos agora gerar a matriz de confusão e a acurácia para os dados de teste, com comandos bem mais simples do que os que

utilizamos até então:

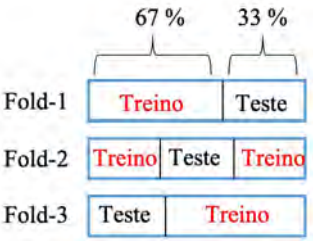
```
> resultado <- confusionMatrix(classesTeste, y[-baseTreino])

> # exibe a matriz de confusão
> resultado$table
##            Reference
## Prediction   N   S
##          N 199  50
##          S  61 210

> # exibe a acurácia
> resultado$overall[1]
##  Accuracy
## 0.7865385
```

A seguir, vamos mostrar na prática como utilizar a técnica de validação cruzada (já explicada anteriormente) para termos maior confiança nos resultados do nosso modelo aplicado em dados ainda não vistos. Vale a pena ressaltar que esta técnica pode (e deve) ser utilizada em qualquer modelo, e não apenas no Naïve Bayes. Como exercício, recomendamos que você tente aplicá-la nos demais modelos deste capítulo.

A técnica de validação cruzada que será demonstrada é a **3-fold-cv**, que consiste em dividir aleatoriamente o conjunto de dados com N elementos em 3 subconjuntos disjuntos (folds) com aproximadamente o mesmo número de elementos (N/3). Cada um dos 3 subconjuntos é usado como conjunto de teste e os restantes são reunidos em um conjunto de treino. O processo é repetido 3 vezes, sendo gerados e avaliados 3 modelos de conhecimento. De forma análoga, são muito comuns as técnicas **5-fold-cv** e **10-fold-cv**, que utilizam, respectivamente 5 e 10 subconjuntos, 5 e 10 repetições do processo, e 5 e 10 modelos gerados.

As figuras a seguir ilustram a validação cruzada 3-fold.

Primeiramente, embaralhamos os índices e dividimos os dados em 3 partições para treino e teste do modelo:




| Índice | Nome | ... |
|---|---|---|
| 1 | Adriano | ... |
| 2 | Mateus | ... |
| 3 | Thiago | ... |
| 4 | João | ... |
| 5 | Pedro | ... |
| 6 | Lucas | ... |
| 7 | Paulo | ... |
| 8 | Timóteo | ... |
| 9 | Maria | ... |
| 10 | Martha | ... |
| 11 | Madalena | ... |
| 12 | Ester | ... |

1º Passo: Embaralhar os Índices

| Índice | Nome | ... |
|---|---|---|
| 9 | Maria | ... |
| 5 | Pedro | ... |
| 6 | Lucas | ... |
| 8 | Timóteo | ... |
| 12 | Ester | ... |
| 1 | Adriano | ... |
| 10 | Martha | ... |
| 2 | Mateus | ... |
| 7 | Paulo | ... |
| 11 | Madalena | ... |
| 3 | Thiago | ... |
| 4 | João | ... |

Os dados ficaram divididos da seguinte forma:

| Índice | Nome | ... | Índice* | Pasta 1 |
|---|---|---|---|---|
| 9 | Maria | ... | 1 | Treino |
| 5 | Pedro | ... | 2 | Treino |
| 6 | Lucas | ... | 3 | Treino |
| 8 | Timóteo | ... | 4 | Treino |
| 12 | Ester | ... | 5 | Treino |
| 1 | Adriano | .. | 6 | Treino |
| 10 | Martha | .. | 7 | Treino |
| 2 | Mateus | ... | 8 | Treino |
| 7 | Paulo | ... | 9 | Teste |
| 11 | Madalena | ... | 10 | Teste |
| 3 | Thiago | ... | 11 | Teste |
| 4 | João | ... | 12 | Teste |

**Pasta 1 de 3**

Treinamento = 8

Teste = 4

| Índice | Nome | ... | Índice* | Pasta 2 |
|---|---|---|---|---|
| 9 | Maria | ... | 1 | Treino |
| 5 | Pedro | ... | 2 | Treino |
| 6 | Lucas | ... | 3 | Treino |
| 8 | Timóteo | ... | 4 | Treino |
| 12 | Ester | ... | 5 | Teste |
| 1 | Adriano | .. | 6 | Teste |
| 10 | Martha | .. | 7 | Teste |
| 2 | Mateus | ... | 8 | Teste |
| 7 | Paulo | ... | 9 | Treino |
| 11 | Madalena | ... | 10 | Treino |
| 3 | Thiago | ... | 11 | Treino |
| 4 | João | ... | 12 | Treino |

**Pasta 2 de 3**

Treinamento = 4

Teste = 4

Treinamento = 4

| Índice | Nome | ... | Índice* | Pasta 3 |
|---|---|---|---|---|
| 9 | Maria | ... | 1 | Teste |
| 5 | Pedro | ... | 2 | Teste |
| 6 | Lucas | ... | 3 | Teste |
| 8 | Timóteo | ... | 4 | Teste |
| 12 | Ester | ... | 5 | Treino |
| 1 | Adriano | .. | 6 | Treino |
| 10 | Martha | .. | 7 | Treino |
| 2 | Mateus | ... | 8 | Treino |
| 7 | Paulo | ... | 9 | Treino |
| 11 | Madalena | ... | 10 | Treino |
| 3 | Thiago | ... | 11 | Treino |
| 4 | João | ... | 12 | Treino |

**Pasta 3 de 3**

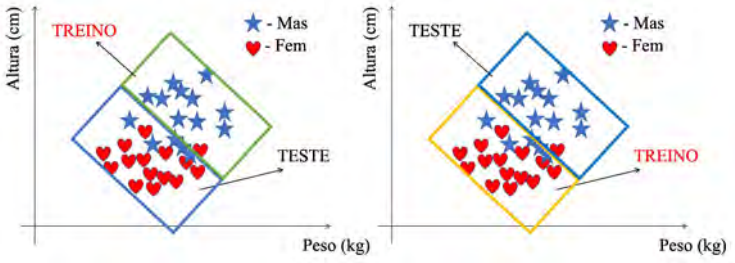
Teste = 4

Treinamento = 8

Finalmente, calculamos a acurácia média das 3 partições:

| Pasta | Acurácia | Exemplo |
|---|---|---|
| 1 | $AccPasta_1$ | 0,65 |
| 2 | $AccPasta_2$ | 0,70 |
| 3 | $AccPasta_3$ | 0,75 |
| Final | $AccTeste$ | $\frac{(0,65+0,70+0,75)}{3} = 0,70$ |

Um problema da validação cruzada k-fold tradicional é a possibilidade da geração de conjuntos de treino e teste desequilibrados, como ilustra a figura a seguir:




Para resolver isso, a validação cruzada k-fold pode ser também estratificada, com o diferencial de que, ao gerar os subconjuntos, a proporção de exemplos em cada uma das classes é considerada durante a amostragem. As figuras a seguir ilustram a geração das partições usando esta técnica:

| Índice | Nome | Peso (kg) | Altura (cm) | Sexo |
|---|---|---|---|---|
| 1 | Adriano | 80 | 185 | M |
| 2 | Mateus | 67 | 175 | M |
| 3 | Thiago | 58 | 187 | M |
| 4 | João | 82 | 176 | M |
| 5 | Pedro | 74 | 168 | M |
| 6 | Lucas | 78 | 174 | M |
| 7 | Paulo | 71 | 172 | M |
| 8 | Timóteo | 92 | 179 | M |
| 9 | Maria | 54 | 167 | F |
| 10 | Martha | 58 | 170 | F |
| 11 | Madalena | 62 | 171 | F |
| 12 | Ester | 57 | 164 | F |

8 Homens = 67% da Base de Dados

4 Mulheres = 33% da Base de Dados

| Índice | Nome | ... |
|---|---|---|
| 1 | Adriano | ... |
| 2 | Mateus | ... |
| 3 | Thiago | ... |
| 4 | João | ... |
| 5 | Pedro | ... |
| 6 | Lucas | ... |
| 7 | Paulo | ... |
| 8 | Timóteo | ... |
| 9 | Maria | ... |
| 10 | Martha | ... |
| 11 | Madalena | ... |
| 12 | Ester | ... |

<u>1º Passo</u>: Embaralhar os Índices de Forma Estratificada

| Índice | Nome | ... |
|---|---|---|
| 5 | Pedro | ... |
| 4 | João | ... |
| 8 | Timóteo | ... |
| 3 | Thiago | ... |
| 6 | Lucas | ... |
| 7 | Paulo | ... |
| 1 | Adriano | ... |
| 2 | Mateus | ... |
| 11 | Madalena | ... |
| 9 | Maria | ... |
| 12 | Ester | ... |
| 10 | Martha | ... |

| Índice | Nome | ... | Índice* | Pasta 1 |
|---|---|---|---|---|
| 5 | Pedro | ... | 1 | Treino |
| 4 | João | ... | 2 | Treino |
| 8 | Timóteo | ... | 3 | Treino |
| 3 | Thiago | ... | 4 | Treino |
| 6 | Lucas | ... | 5 | Treino |
| 7 | Paulo | .. | 6 | Teste |
| 1 | Adriano | .. | 7 | Teste |
| 2 | Mateus | ... | 8 | Teste |
| 11 | Madalena | ... | 9 | Treino |
| 9 | Maria | ... | 10 | Treino |
| 12 | Ester | ... | 11 | Treino |
| 10 | Martha | ... | 12 | Teste |

**<u>Pasta 1 de 3</u>**

Treinamento = 8
5 M's e 3 F's

Teste = 4
3 M's e 1 F

| Índice | Nome | ... | Índice* | Pasta 2 |
|---|---|---|---|---|
| 5 | Pedro | ... | 1 | Treino |
| 4 | João | ... | 2 | Treino |
| 8 | Timóteo | ... | 3 | Teste |
| 3 | Thiago | ... | 4 | Teste |
| 6 | Lucas | ... | 5 | Teste |
| 7 | Paulo | .. | 6 | Treino |
| 1 | Adriano | .. | 7 | Treino |
| 2 | Mateus | ... | 8 | Treino |
| 11 | Madalena | ... | 9 | Treino |
| 9 | Maria | ... | 10 | Treino |
| 12 | Ester | ... | 11 | Teste |
| 10 | Martha | ... | 12 | Treino |

**Pasta 2 de 3**

Treinamento = 8
5 M's e 3 F's

Teste = 4
3 M's e 1 F

| Índice | Nome | ... | Índice* | Pasta 3 |
|---|---|---|---|---|
| 5 | Pedro | ... | 1 | Teste |
| 4 | João | ... | 2 | Teste |
| 8 | Timóteo | ... | 3 | Treino |
| 3 | Thiago | ... | 4 | Treino |
| 6 | Lucas | ... | 5 | Treino |
| 7 | Paulo | .. | 6 | Treino |
| 1 | Adriano | .. | 7 | Treino |
| 2 | Mateus | ... | 8 | Treino |
| 11 | Madalena | ... | 9 | Teste |
| 9 | Maria | ... | 10 | Teste |
| 12 | Ester | ... | 11 | Treino |
| 10 | Martha | ... | 12 | Treino |

**Pasta 3 de 3**

Treinamento = 8
6 M's e 2 F's

Teste = 4
2 M's e 2 F's

Neste exemplo prático, utilizaremos a técnica **3-fold-cv estratificada**. Primeiro, criamos as partições:

```
> particoes <- createFolds(tabela$ADIMPLENTE, k=3)
```

Se quisermos visualizar os elementos pertencentes a cada uma das partições, podemos utilizar o comando `particoes` , ou para visualizar apenas os elementos pertencentes à primeira, segunda ou terceira partição, podemos utilizar os comandos `particoes$Fold1` , `particoes$Fold2` e `particoes$Fold3` ,

respectivamente.

Em seguida, fazemos as divisões dos dados em treino e teste. Sempre treinaremos o modelo com duas partições e o testaremos com a terceira partição restante. Para tal, vamos criar as seguintes variáveis, representando cenários de treino e teste:

- `tabTeste1` , com a partição 1
- `tabTreino1` , com as partições 2 e 3
- `tabTeste2` , com a partição 2
- `tabTreino2` , com as partições 1 e 3
- `tabTeste3` , com a partição 3
- `tabTreino3` , com as partições 1 e 2

```
> tabTeste1 <- tabela[particoes$Fold1, ]
> tabTreino1 <- tabela[-particoes$Fold1, ]

> tabTeste2 <- tabela[particoes$Fold2, ]
> tabTreino2 <- tabela[-particoes$Fold2, ]

> tabTeste3 <- tabela[particoes$Fold3, ]
> tabTreino3 <- tabela[-particoes$Fold3, ]
```

Agora, vamos criar um modelo para cada um dos cenários de treino (variáveis `tabTeste1` , `tabTeste2` e `tabTeste3` ), e tentar prever a situação do cliente apenas em função da idade (em vez de utilizar todos os atributos, como fizemos anteriormente):

```
> NB1 <- naiveBayes(ADIMPLENTE~IDADE, data = tabTreino1)
> NB2 <- naiveBayes(ADIMPLENTE~IDADE, data = tabTreino2)
> NB3 <- naiveBayes(ADIMPLENTE~IDADE, data = tabTreino3)
```

Vamos então realizar as predições para as bases de teste `tabTeste1` , `tabTeste2` e `tabTeste3` :

```
> PREDNB1 <- predict(NB1, newdata = tabTeste1)
> PREDNB2 <- predict(NB2, newdata = tabTeste2)
```

```
> PREDNB3 <- predict(NB3, newdata = tabTeste3)
```

Em seguida, calculamos as métricas de avaliação (matriz de confusão e acurácia) para cada modelo de uma forma diferente:

```
> MATCONFNB1 <- table(PREDNB1, tabTeste1$ADIMPLENTE, deparse.leve
l = 2)
> MATCONFNB2 <- table(PREDNB2, tabTeste2$ADIMPLENTE, deparse.leve
l = 2)
> MATCONFNB3 <- table(PREDNB3, tabTeste3$ADIMPLENTE, deparse.leve
l = 2)

> show(MATCONFNB1)
##        tabTeste1$ADIMPLENTE
## PREDNB1   N   S
##       N 247 153
##       S  81 211

> show(MATCONFNB2)
##        tabTeste2$ADIMPLENTE
## PREDNB2   N   S
##       N 223 138
##       S 106 225

> show(MATCONFNB3)
##        tabTeste3$ADIMPLENTE
## PREDNB3   N   S
##       N 243 139
##       S  86 225

> ACC1 <- sum(diag(MATCONFNB1))/nrow(tabTeste1)
> ACC2 <- sum(diag(MATCONFNB2))/nrow(tabTeste2)
> ACC3 <- sum(diag(MATCONFNB3))/nrow(tabTeste3)
```

Finalmente, calculamos a acurácia final, que é a média das acurácias dos 3 modelos:

```
> ACCFINAL <- ( ACC1 + ACC2 + ACC3 ) / 3
> ACCFINAL*100
## [1] 66.15244
```

Caso quiséssemos realizar o treinamento em função de todas as

variáveis (em vez de apenas a variável `Idade`) faríamos, na etapa de construção do modelo:

```
> NB1 <- naiveBayes(ADIMPLENTE~ ., data = tabTreino1)
> NB2 <- naiveBayes(ADIMPLENTE~ ., data = tabTreino2)
> NB3 <- naiveBayes(ADIMPLENTE~ ., data = tabTreino3)
```

O restante do código seria exatamente igual ao anterior.

## 6.4 SUPPORT VECTOR MACHINE (SVM)

Para ilustrar o algoritmo SVM em um problema de Classificação, vamos usar a mesma base de dados em Excel que utilizamos para o Naïve Bayes (este arquivo está disponível em https://github.com/tatianaesc/introdatascience/). Para executar esta parte prática, copie os comandos a seguir no seu R Script e execute (ou baixe o script diretamente do GitHub).

O início do script é exatamente igual ao que fizemos na prática do Naïve Bayes: primeiro, vamos importar os dados; depois, armazenamos os atributos no objeto `x` e as classes no objeto `y`; finalmente, sorteamos 75% dos dados para compor a base de treino, usando uma semente para garantir a reprodutibilidade dos resultados deste script:

```
> tabela = read.table("BD_aulaclass.csv", sep = ";", dec = ",", h
eader = TRUE)
> summary(tabela)

> x <- tabela
> x$ADIMPLENTE <- NULL
> y <- tabela$ADIMPLENTE

> set.seed(2018)
> N = nrow(x)
> baseTreino <- sample(1:N, N*0.75, FALSE)
```

Para possibilitar a construção do modelo do SVM, será necessário instalar e carregar os pacotes **kernlab** e **mlbench**, que incluem as bibliotecas para este modelo.

```
> install.packages("kernlab")
> install.packages("mlbench")
> library(kernlab)
> library(mlbench)
```

A seguir, vamos construir o modelo para prever `y` em função de `x`, usando a base de treino e kernel gaussiana, representada pelo valor `rbfdot` no parâmetro `kernel`:

```
> modeloSVM <- ksvm(y[baseTreino]~., data=x[baseTreino,], kernel=
"rbfdot")
> print(modeloSVM)
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.121695743898903
##
## Number of Support Vectors : 661
##
## Objective Function Value : -471.8287
## Training error : 0.100193
```

Vamos então aplicar o modelo nos dados de teste:

```
> classesTeste <- predict(modeloSVM, x[-baseTreino,], type="respo
nse")
```

Para avaliar o modelo nos dados de teste, vamos novamente utilizar o pacote **caret** para construir a matriz de confusão e exibir a acurácia. Assumindo que o pacote já está instalado, basta carregá-lo:

```
>  library("caret")
```

```
> resultado <- confusionMatrix(classesTeste, y[-baseTreino])
> resultado$table
##           Reference
## Prediction   N   S
##          N 196   1
##          S  64 259

> resultado$overall[1]
## Accuracy
##    0.875
```

Vamos verificar como seria a acurácia de um modelo SVM construído um tipo diferente de função kernel, a polinomial ( `polydot` ) em vez de gaussiana. Primeiro, construímos o modelo com os dados de treino:

```
>  modeloSVM <- ksvm(y[baseTreino]~., data=x[baseTreino,], kernel=
"polydot")
> print(modeloSVM)
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Polynomial kernel function.
##  Hyperparameters : degree =  1  scale =  1  offset =  1
##
## Number of Support Vectors : 500
##
## Objective Function Value : -489.1002
## Training error : 0.102119
```

Em seguida, aplicamos e avaliamos o modelo nos dados de teste:

```
> classesTeste <- predict(modeloSVM, x[-baseTreino,], type="respo
nse")

> resultado <- confusionMatrix(classesTeste, y[-baseTreino])
> resultado$table
##           Reference
## Prediction   N   S
##          N 209   5
```

```
##          S  51 255
```

```
> resultado$overall[1]
##  Accuracy
## 0.8923077
```

## Resumo

Este capítulo apresentou conceitos práticos dos modelos de aprendizagem supervisionada que podem ser usados para resolver problemas de Classificação apresentados no capítulo anterior.

Capítulo 7

# MODELOS DE REGRESSÃO

Os problemas de Classificação, estudados no capítulo anterior, podem ser considerados um subtipo dos problemas de Regressão (também conhecidos como problemas de estimação), pois funcionam de forma similar, com a diferença de que, em vez de o resultado ser categórico, como é na Classificação, o resultado é numérico (contínuo ou discreto). Por exemplo, um problema de Classificação seria "Conceder ou não crédito para um cliente?" enquanto um problema de Regressão seria "Conceder qual valor de crédito para um cliente?". As tarefas como: preparação da base de dados, separação em conjuntos de treino e teste, definição dos critérios de parada do algoritmo, treinamento e teste são feitas de forma equivalente para ambos os problemas. Este capítulo detalhará os problemas de Regressão.

## 7.1 PROBLEMAS DE REGRESSÃO

Assim como na Classificação, a Regressão consiste em realizar aprendizado supervisionado a partir de dados históricos. Além do tipo do resultado, a principal diferença entre os dois problemas está na avaliação de saída: na Regressão, em vez de se estimar a **acurácia**, estima-se a **distância** ou o **erro** entre a saída do estimador (modelo) e a saída desejada. A saída de um estimador é

um valor numérico contínuo que deve ser o mais próximo possível do valor desejado, e a diferença entre esses valores fornece uma medida de erro de estimação do algoritmo.

Para entender em linhas gerais o que é um problema de Regressão, considere um grupo varejista com esta região de negócios:

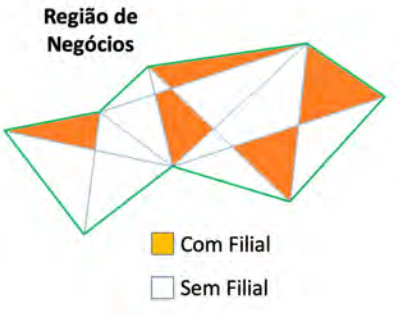



Figura 7.1: Exemplo de distribuição de filiais de uma rede varejista

Este grupo deseja expandir suas filiais, mas tem o seguinte problema: onde abrir uma nova filial? A resposta para esta pergunta pode ser determinada usando como métrica seu Faturamento Médio Anual. Esta métrica é conhecida nas regiões onde o grupo já tem filiais, mas desconhecida nas regiões onde não tem.

Um problema de Regressão seria estimar estes valores de regiões em potencial para novas filiais. Para tal, o primeiro passo seria levantar variáveis que estão tanto presentes nos bairros com e sem filiais, tais como: renda per capita, IDH, número de concorrentes, número de habitantes, preço do $m^2$ etc. A seguir, separamos nossos dados em dois conjuntos: com e sem

faturamento, como ilustra a tabela a seguir:

| | Bairro | Renda/Hab. | ... | Preço do $m^2$ | Faturamento |
|---|---|---|---|---|---|
| Com | A | 1500 | ... | 2500 | 105.000 |
| | H | 2400 | ... | 5300 | 180.000 |
| | ... | ... | ... | ... | ... |
| | L | 3400 | ... | 2750 | 150.000 |
| Sem | B | 2500 | ... | 1780 | NA |
| | C | 1000 | ... | 3500 | NA |
| | D | 4300 | ... | 6500 | NA |
| | ... | ... | ... | ... | ... |
| | K | 7000 | ... | 8900 | NA |
| | M | 2800 | ... | 3900 | NA |

Com estes dados em mãos, basta elaborarmos um modelo de Regressão para obter os valores estimados para as regiões sem filiais:

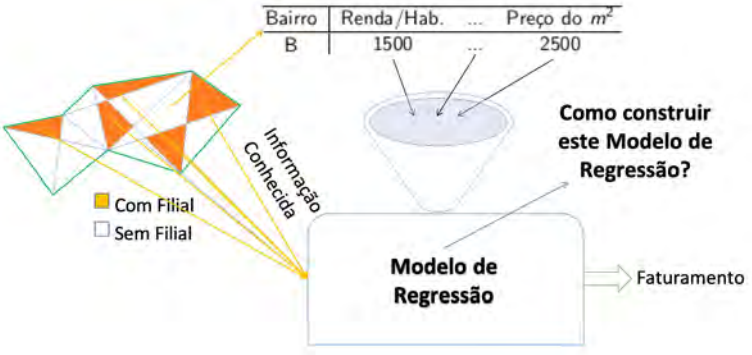



Figura 7.3: Aplicação do modelo de Regressão no problema

Finalmente, a partir do modelo, seleciona-se o bairro com Maior Faturamento Esperado para abrir uma nova filial:

| | Bairro | Renda/Hab. | ... | Preço do $m^2$ | Faturamento |
|---|---|---|---|---|---|
| Observado | A | 1500 | ... | 2500 | 105.000 |
| | H | 2400 | ... | 5300 | 180.000 |
| | ... | ... | ... | ... | ... |
| | L | 3400 | ... | 2750 | 150.000 |
| Esperado | B | 2500 | ... | 1780 | 125.000 |
| | **C** | **1000** | ... | **3500** | **200.000** |
| | D | 4300 | ... | 6500 | |
| | ... | ... | ... | ... | ... |
| | K | 7000 | ... | 8900 | 180.000 |
| | M | 2800 | ... | 3900 | 120.000 |

Assim, podemos definir um problema de Regressão como: dado um conjunto de $n$ padrões, onde cada um é composto por informação de variáveis explicativas (independentes) e de uma variável resposta contínua ou discreta (dependente), tem-se como objetivo construir um modelo de Regressão que, dado um novo padrão, estime o **valor mais esperado** para a variável resposta. A tabela a seguir ilustra o problema de Regressão:

| Padrão | Explicativas | | | Resposta |
|---|---|---|---|---|
| $\mathbf{x}_1$ | $x_{11}$ | ... | $x_{1J}$ | $y_1$ |
| $\mathbf{x}_2$ | $x_{21}$ | ... | $x_{2J}$ | $y_2$ |
| $\mathbf{x}_3$ | $x_{31}$ | ... | $x_{3J}$ | $y_3$ |
| ... | ... | ... | ... | ... |
| $\mathbf{x}_i$ | $x_{i1}$ | ... | $x_{iJ}$ | $y_i$ |
| ... | ... | ... | ... | ... |
| $\mathbf{x}_n$ | $x_{n1}$ | ... | $x_{nJ}$ | $y_n$ |

Formalmente, seja $d_j = 1, ..., n$ a resposta desejada para o objeto $j$ e $y_j$ a resposta predita do algoritmo, obtida a partir da entrada $x_j s$.

Então, $e_j = d_j - y_j$ é o erro observado na saída do sistema para o objeto $j$. O processo de treinamento do estimador tem por objetivo corrigir este erro observado e, para tal, busca minimizar um

critério (função objetivo) baseado em $e_j$, de maneira que os valores de $y_j$ estejam próximos dos de $d_j$ no sentido estatístico.

Existem diversas métricas de desempenho para problemas de Regressão, sendo uma das mais usadas a RMSE (*Root Mean Squared Error*, ou raiz do erro quadrático médio). Quanto menor o RMSE, melhor é o modelo de Regressão analisado e sua fórmula é dada por:

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n} e_j^2}$$

Outra métrica muito utilizada é o Coeficiente de Determinação, ou $R^2$. Quanto mais próximo de 1, melhor o ajuste do modelo de Regressão aos dados e sua fórmula é dada por:

$$R^2 = 1 - \frac{SQE}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2}$$

Sendo SQE a Soma dos quadrados dos erros (*Sum of Squared Errors*), calculada por:

$$SQE = \sum_{j=1}^{n} e_j^2$$

Vale a pena mencionar o teorema "não existe almoço grátis":

não existe um algoritmo de aprendizado que seja superior a todos os demais quando considerados todos os problemas possíveis. A cada problema, os algoritmos disponíveis devem ser experimentados a fim de identificar aqueles que obtêm melhor desempenho.

## 7.2 ALGORITMOS

A seguir, serão apresentados alguns algoritmos que podem ser usados para problemas de Regressão. Lembramos que tarefas como preparação da base de dados, separação em conjuntos de treino e teste, definição dos critérios de parada do algoritmo, treinamento e teste são feitas de forma equivalente, tanto em problemas de Classificação quanto Regressão.

### Árvore de Decisão

Quando utilizadas para problemas de Regressão, as Árvores de Decisão são chamadas Árvores de Regressão. Foram introduzidas nos anos 1980 como parte do algoritmo CART (*Classification and Regression Tree*). São muito similares às Árvores de Classificação, com a diferença de que, nas Árvores de Regressão, a predição (ou valor estimado) é a média dos valores dos exemplos de cada folha.

Também são construídas de forma similar às Árvores de Classificação: a partir do nó raiz, os dados são particionados usando uma estratégia de divisão e conquista de acordo com a característica que resultará no resultado mais homogêneo após a separação ser realizada. Enquanto nas Árvores de Classificação a homogeneidade é medida pela entropia, nas Árvores de Regressão, a homogeneidade é medida por estatísticas como variância, desvio

padrão ou desvio absoluto da média.

Um critério de divisão comum para Árvores de Regressão é a redução de desvio padrão (SDR – *Standard Deviation Reduction*). Esta fórmula mede a redução no desvio padrão, comparando o desvio padrão antes da divisão com o desvio padrão ponderado após a divisão:

$$\text{SDR} = sd(T) - \sum_i \frac{|T_i|}{|T|} \times sd(T_i)$$

A função *sd(T)* refere-se ao desvio padrão dos valores no conjunto *T*, enquanto $T_1$, $T_2$, ..., $T_n$, são os conjuntos de valores resultantes de uma divisão em uma característica. $|T|$ refere-se ao número de observações no conjunto *T*.

A figura a seguir ilustra uma Árvore de Regressão:

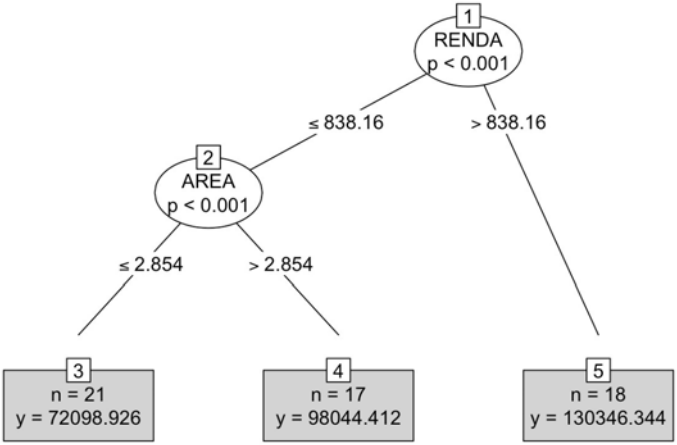



Figura 7.10: Exemplo de Árvore de Regressão

## KNN

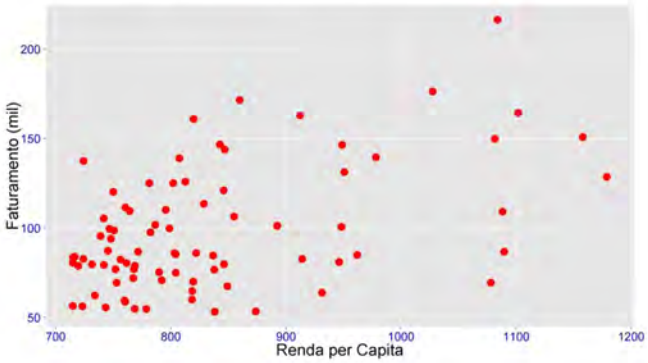
O KNN também pode ser usado para problemas de Regressão: neste caso, o valor estimado é a média aritmética dos *k*-vizinhos mais próximos. As perguntas sobre qual o valor de *k* adequado e qual métrica de distância usar também são aplicáveis para este tipo de problema.

## Regressão Linear

A Regressão Linear, como o nome já diz, é um algoritmo para o problema de Regressão. Vamos voltar ao exemplo dado anteriormente, da escolha de um bairro para abrir uma nova filial para apresentar o algoritmo de Regressão Linear. Vamos olhar primeiro apenas para os dados que possuem valores para a variável

resposta e, para simplificar, vamos somente considerar as variáveis "Renda per Capita" e "Faturamento". A figura a seguir ilustra os dados que estamos trabalhando, tanto em formato tabular quanto distribuídos no espaço.

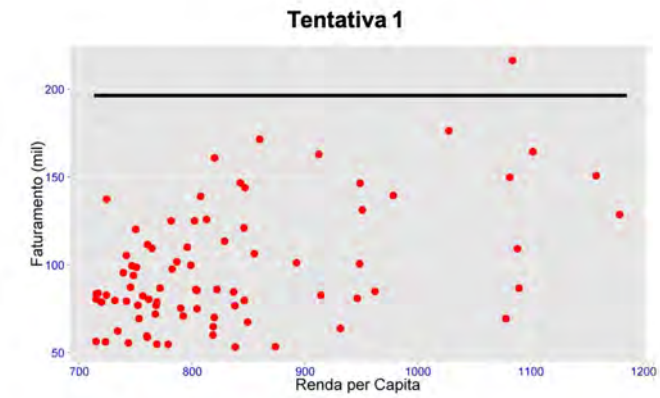
| Bairro | Renda/Hab. | Faturamento |
|---|---|---|
| A | 1500 | 105.000 |
| H | 2400 | 180.000 |
| ... | ... | ... |
| L | 3400 | 150.000 |




É fácil observar que quanto maior a renda per capita do bairro, maior tende a ser o Faturamento no Bairro. Mas como formular essa relação matematicamente? A Regressão Linear traz a solução: vamos ajustar uma reta que melhor passe pelos pontos, ou seja:

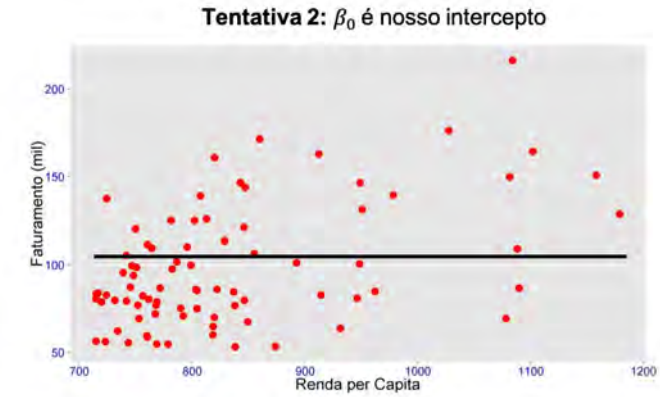
$$\widehat{Fat}_{Bairro} = \beta_0 + \beta_1 \times RendaPerCapita_{Bairro}$$

Vamos começar com a primeira tentativa, fazendo uma reta simples:




**Modelo:** $\widehat{Fat}_{Bairro} = 190$

Matematicamente, o faturamento estimado para todos os bairros é de 190, o que não parece um valor muito real. Vamos tentar novamente, alterando o valor do intercepto (onde a linha corta o eixo y):

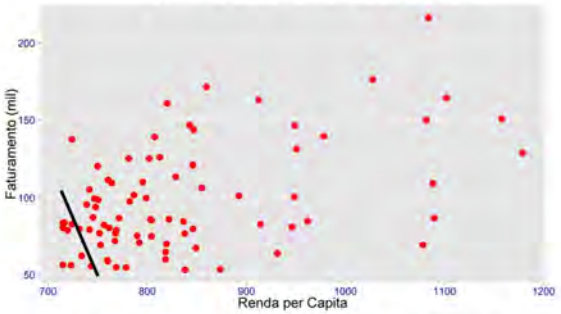



**Modelo:** $\widehat{Fat}_{Bairro} = 120$

Agora, o faturamento estimado para todos os bairros é de 120, apesar de a reta estar um pouco mais próxima dos pontos, este modelo também não parece muito útil. Vamos então tentar usar a informação de renda per capita, diminuindo, por exemplo, 2 vezes o seu valor pelo faturamento estimado que temos até o momento:

Percebemos que agora a reta sumiu do gráfico. Vamos então corrigir o intercepto para ver se obtemos melhores resultados:

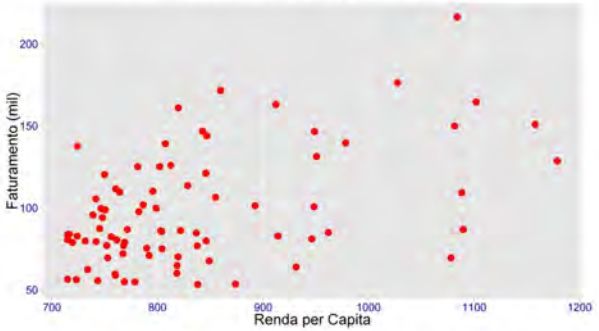
**Tentativa 4:** Corrigindo o intercepto




**Modelo:** $\widehat{Fat}_{Bairro} = 1502 - 2 \times RendaPerCapita_{Bairro}$

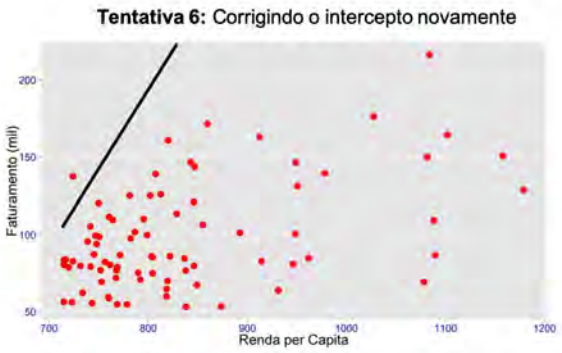
Mas espere... Tínhamos observado anteriormente que a relação estre as variáveis renda per capita e faturamento era positiva. Vamos então trocar o sinal da nossa equação para exprimir essa relação:

**Tentativa 5:** Onde foi parar a reta novamente?

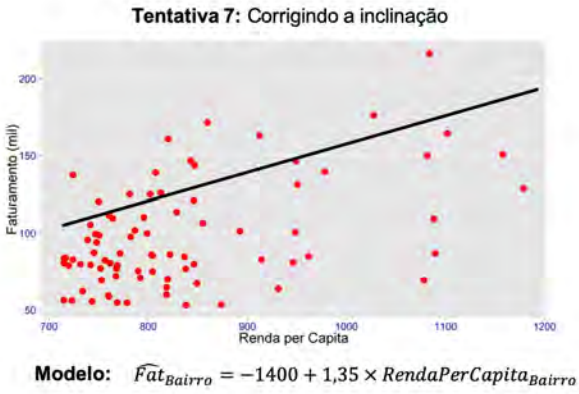



**Modelo:** $\widehat{Fat}_{Bairro} = 1502 + 2 \times RendaPerCapita_{Bairro}$

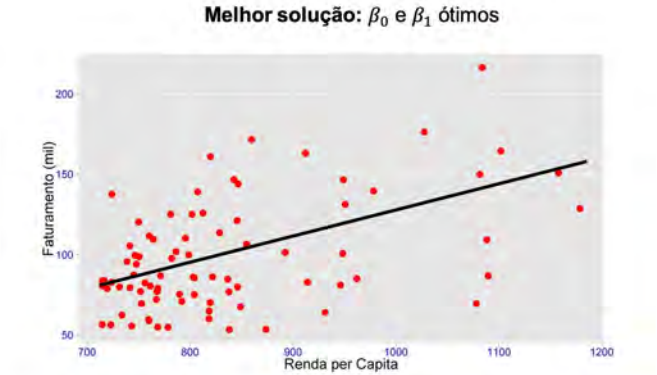
A reta sumiu novamente! Vamos, outra vez, corrigir o intercepto:



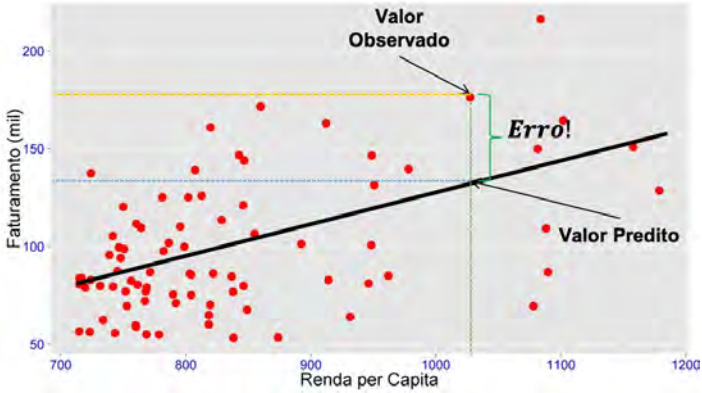

Estamos chegando perto, bastando corrigir um pouco a inclinação da reta:



Se continuarmos neste processo manual (e cansativo!), vamos em algum momento chegar à solução ótima para o problema, ilustrada na figura a seguir:



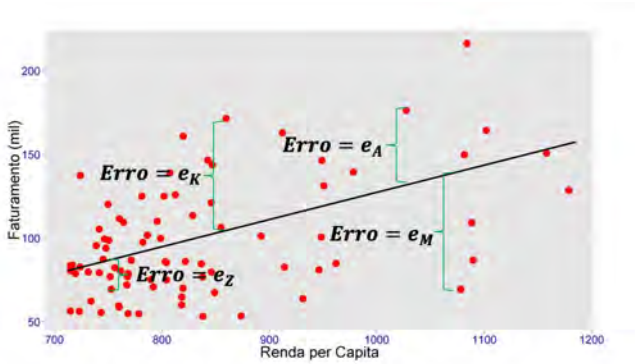

Este modelo significa que, a cada aumento de R$100 na renda per capita do bairro, espera-se que isso reflita em $0{,}15 * 100 = 15$ mil de faturamento para a filial. Esta solução é dita ótima porque passa mais perto dos pontos (considerando a distância euclidiana):



Logo, para cada escolha dos parâmetros $\beta_0$ e $\beta_1$ na equação

$$\widehat{Fat}_{Bairro} = \beta_0 + \beta_1 \times RendaPerCapita_{Bairro}$$

é possível calcular os erros (ou desvios) dessa escolha. Porém, observe que para este modelo:




$$\widehat{Fat}_{Bairro} = -24{,}49 + 0{,}15 \times RendaPerCapita_{Bairro}$$

se somarmos os erros para calcular o erro total do modelo, como os erros individuais são positivos e negativos, eles se anulam:

| Bairro | $Fat$ | $\widehat{Fat}$ | $e = Fat - \widehat{Fat}$ |
|---|---|---|---|
| A | 180 | 130 | 50 |
| K | 175 | 115 | 60 |
| M | 65 | 120 | -55 |
| ... | ... | ... | ... |
| Z | 70 | 82 | -12 |

Assim, a melhor prática é trabalhar com a **magnitude do erro**

(erro ao quadrado, por exemplo):

$$\sum_{i \in Bairro} e_i^2 = (Fat_i - \hat{Fat}_i)^2 = 50^2 + 60^2 + (-55)^2 + ... + (-12)^2 = 230$$

Moral da história: a Regressão Linear consiste em escolher $\beta_0$ e $\beta_1$ para construir uma reta que minimize a soma dos quadrados dos erros (SQE):

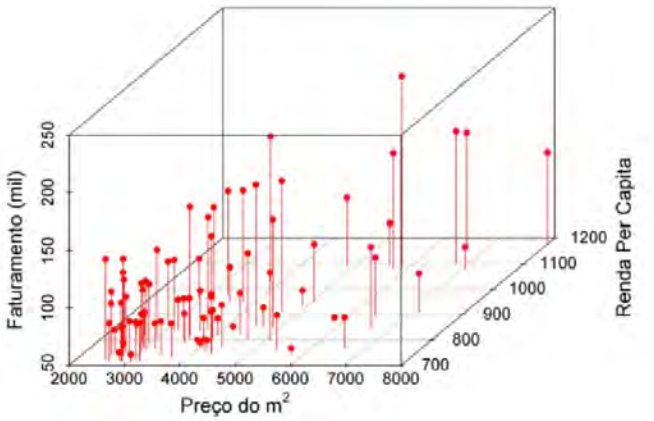
$$SQE = \sum_{i \in Bairro}^{n} e_i^2$$

Na primeira tentativa, o SQE foi 3224, na segunda, 1224, e assim por diante, até que chegamos à melhor solução, com SQE 230.

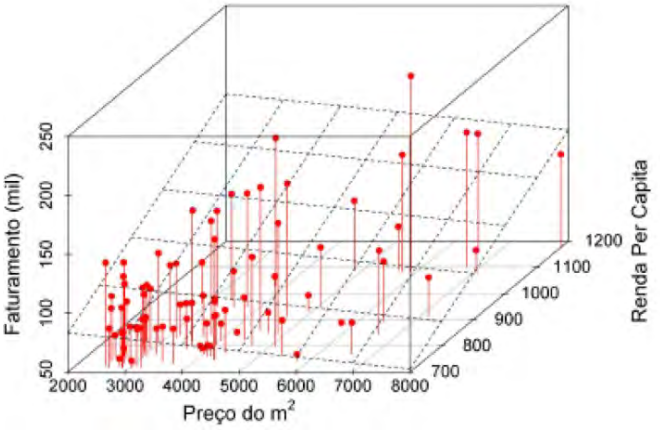
Vale a pena ressaltar que neste exemplo consideramos apenas a relação entre faturamento e renda per capita, mas, em problemas reais, dificilmente haverá uma única variável x capaz de prever a saída y. Se quiséssemos adicionar uma nova variável (como preço do $m^2$), teríamos o modelo:

$$\widehat{Fat}_i = \beta_0 + \beta_1 \times RendaPerCapita_i + \beta_2 \times PrecoM^2$$

Este problema seria representado graficamente como:




E a solução seria construir um plano (em vez de uma reta) que melhor se ajuste aos pontos:




Neste caso, dizemos que temos uma Regressão Linear **múltipla**. Basta adicionar ao modelo as demais variáveis preditoras ($x_1$, $x_2$, ..., $x_n$) e seus coeficientes correspondentes, que podem ser

estimados pelo Método dos Mínimos Quadrados (*Ordinary Least Squares*). Este método busca encontrar o melhor ajuste para um conjunto de dados tentando minimizar a soma dos quadrados das diferenças entre o valor estimado e os dados observados. Para entender mais sobre este método, consulte https://pt.wikipedia.org/wiki/M%C3%A9todo_dos_m%C3%ADnimos_quadrados e https://towardsdatascience.com/linear-regression-understanding-the-theory-7e53ac2831b5.

Formalmente, a Regressão Linear modela a relação entre a variável de resposta ($y$) e as variáveis preditoras ($X$). A Regressão corresponde ao problema de estimar uma função a partir de pares entrada-saída e considera que $y$ pode ser explicado por uma combinação linear de $X$. Quando temos apenas um $x$ em $X$, temos uma Regressão Linear simples, cuja equação é $\mathbf{y = \beta_0 + \beta_1 x}$, sendo $\beta_0$ e $\beta_1$ os coeficientes de regressão (especificam, respectivamente, o intercepto do eixo y e a inclinação da reta). Para o caso da Regressão Linear múltipla, a equação deve ser estendida para equação de plano/hiperplano: $\mathbf{y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + + \beta_n x_n}$.

Ou seja, a solução da tarefa de regressão consiste em encontrar valores para os coeficientes de regressão de forma que a reta (ou plano/hiperplano) se ajuste aos valores assumidos pelas variáveis no conjunto de dados. Além do Método dos Mínimos Quadrados, existem diversas técnicas matemáticas para determinar os coeficientes, que estão fora do escopo deste livro, mas caso você tenha interesse, podem ser consultadas em https://pt.wikipedia.org/wiki/Regress%C3%A3o_linear/.

A saída do estimador é um valor numérico contínuo que deve ser o mais próximo possível do valor desejado, e a diferença entre

esses valores fornece uma medida de erro de estimação do algoritmo. Então, seja $d_j = 1, ..., n$ a resposta desejada para o objeto $j$ e $y_j$ a resposta predita do algoritmo, obtida a partir da entrada $x_j$. Então, $e_j = d_j - y_j$ é o erro observado na saída do sistema para o objeto $j$.

O processo de treinamento do estimador tem por objetivo corrigir este erro observado e, para tal, busca minimizar um critério (função objetivo) baseado em $e_j$, de maneira que os valores de $y_j$ estejam próximos dos de $d_j$ no sentido estatístico. Se a equação de regressão aproxima suficiente bem os dados de treinamento, então ela pode ser usada para estimar o valor de uma variável (y) a partir do valor da outra variável (X), assumindo uma relação linear entre as estas variáveis. Em suma, a Regressão Linear procura pelos coeficientes da reta que minimizam a distância dos objetos à reta.

No exemplo apresentado, utilizamos como medida de avaliação a soma dos quadrados dos erros (SQE) que é bem simples de calcular. Porém, na prática, a RMSE (*Root Mean Squared Error*) é mais utilizada. Outra métrica muito utilizada é o $R^2$, ou Coeficiente de Determinação, que utiliza o SQE no seu cálculo, e significa que quanto mais $R^2$ estiver próximo de 1, mais a reta estará bem ajustada aos dados. Estas fórmulas já foram exibidas no início desta seção.

## Regressão Logística

A Regressão Logística, apesar do nome, é um algoritmo utilizado para problemas de Classificação, mas optamos por apresentá-lo neste capítulo porque seu funcionamento lembra

muito o funcionamento do algoritmo de Regressão Linear. O algoritmo da Regressão Logística é usado para estimar valores discretos (valores binários como 0/1, sim/não, verdadeiro/falso) com base em um conjunto de variáveis independentes. Internamente, a Regressão Logística calcula a probabilidade de ocorrência de um evento, ajustando os dados a uma função *logit* (por isso, também é conhecido como regressão *logit*). Esta função pode ser descrita por:

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right) = \log(p) - \log(1-p).$$

Como prevê a probabilidade, seus valores de saída estão entre 0 e 1 (como esperado). Este algoritmo utiliza a função logística, também chamada de sigmóide: uma curva em forma de S que pode mapear qualquer número em um intervalo entre 0 e 1 (mas nunca exatamente nestes limites), dada por:

$$f(x) = \frac{1}{1 + e^{-x}}$$

onde $f(x)$ é a saída prevista, $\beta_0$ é o intercepto e $\beta_1$ é o coeficiente para o atributo $x$, que deve ser aprendido a partir dos dados de treino.

A figura a seguir ilustra uma transformação logística entre -6 e 6:

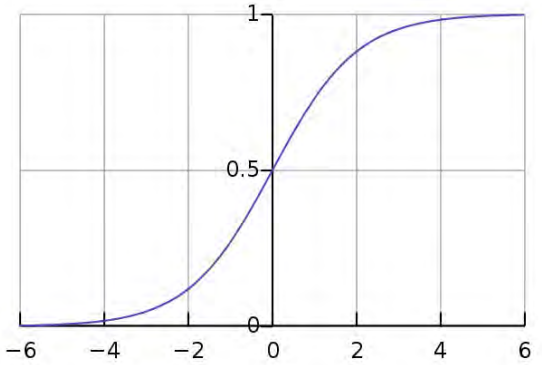



Figura 7.32: Exemplo de transformação logística (Fonte: Wikipedia - https://en.wikipedia.org/wiki/Logistic_function.)

De forma similar à Regressão Linear, a Regressão Logística usa uma equação como representação: os valores de entrada ($x$) são combinados linearmente usando pesos ou valores de coeficiente para prever um valor de saída ($y$). A diferença é que o valor de saída é modelado em valor binário (0 ou 1) em vez de um valor numérico.

A Regressão Logística modela a probabilidade da classe padrão do problema. Por exemplo, se estivermos modelando o sexo de uma pessoa dada sua altura e considerarmos a classe "masculino" como padrão, o modelo pode ser escrito como: **P(sexo = masculino | altura)**.

Escrito de outra forma, estamos modelando a probabilidade de uma entrada X pertencer à classe padrão (Y = 1): P(X) = **P(Y=1 | X)**. Note que a predição da probabilidade pode ser transformada em um valor binário (0 ou 1) para fazer a classificação.

Os coeficientes da Regressão Logística podem ser estimados

usando os dados de treinamento, através do método de estimação de máxima verossimilhança, um método matemático que busca valores para os coeficientes de forma a minimizar o erro nas probabilidades preditas pelo modelo para os dados, cujo detalhamento está fora do escopo deste livro. Os melhores coeficientes resultarão em um modelo que vai prever um valor muito próximo de 1 para a classe padrão e um valor muito próximo de 0 para a outra classe.

Após determinados os coeficientes, para fazer predições com a Regressão Logística, basta calcular os coeficientes e aplicar a equação resultante.

## Resumo

Este capítulo apresentou os conceitos teóricos de alguns modelos de aprendizagem supervisionada que podem ser usados para resolver problemas de Regressão, e também o modelo de Regressão Logística, usado para problemas de Classificação. O próximo capítulo apresentará a parte prática.

Capítulo 8

# PRÁTICAS DE REGRESSÃO

No capítulo anterior, apresentamos os conceitos teóricos de alguns modelos de aprendizagem supervisionada que podem ser usados para resolver problemas de Regressão, e também o modelo de Regressão Logística, usado para problemas de classificação. Este capítulo apresenta exemplos práticos destes modelos. O script deste capítulo está disponível em https://github.com/tatianaesc/introdatascience/.

## 8.1 REGRESSÃO LINEAR, ÁRVORE DE REGRESSÃO E KNN PARA REGRESSÃO

Para exemplificar a utilização da Regressão Linear, da Árvore de Regressão e do KNN para Regressão no R, vamos utilizar o exemplo apresentado na parte teórica, que consiste em escolher um bairro para abrir uma nova filial de uma loja, para estimar o valor de faturamento para os bairros em que ainda não há filial. O melhor candidato para receber uma nova filial será o bairro com maior faturamento esperado.

A base de dados que será utilizada contém as seguintes variáveis:

- `BAIRRO` : id do bairro;

- `POP` : população do bairro;
- `RENDA` : renda média do bairro;
- `ESPVIDA` : esperança de vida da população do bairro;
- `TXALFA` : taxa de alfabetismo;
- `NCONC` : número de concorrentes encontrados;
- `AREA` : área disponível;
- `PRECOM2` : preço do metro quadrado;
- `ROUBOSLOJAS` : número médio de roubos nas lojas;
- `ROUBOSRES` : número médio de roubos nas residências;
- `FAT` : faturamento anual.

Para executar esta parte prática, copie os comandos a seguir no seu R Script e execute (ou baixe o script diretamente de https://github.com/tatianaesc/introdatascience/). Vamos inicialmente ler o arquivo `faturamento.csv` , disponível no repositório do GitHub, que deve estar na mesma pasta do script R:

```
> tabela <- read.table("faturamento.csv", sep = ";",
                       dec = ",", header = TRUE)
```

Em seguida, vamos realizar a separação dos dados importados em duas bases, de treinamento (com valores de faturamento preenchidos) e de teste (sem valores de faturamento):

```
>  # determina os índices com valores faltantes de faturamento
> idxNA <- is.na(tabela$FAT)
> # dados com valores faltantes (NA) de faturamento
> tabelaNA <- tabela[idxNA, ]
# dados sem valores faltantes de faturamento
> tabela <- na.omit(tabela)
```

Agora, vamos construir um gráfico de dispersão com os nossos dados (renda per capita e faturamento). Para tal, será necessário instalar e carregar o pacote **ggplot2**, que possibilita a construção deste gráfico:

```
> install.packages("ggplot2")
> library("ggplot2")
```

O gráfico de dispersão Faturamento x Renda é construído com os seguintes comandos:

```
> estetica = aes(x = RENDA, y = FAT)
> base = ggplot(data = tabela, estetica)
> base + geom_point(size = 4)
```

Para entender melhor como funciona a Regressão Linear, vamos criar uma primeira reta de regressão no gráfico Faturamento x Renda, onde *y* é sempre igual a 1. Para tal, faça:

```
> estetica = aes(x = RENDA, y = FAT)
> base = ggplot(data = tabela, estetica)
> base + geom_point(size = 4) + stat_smooth(method = "lm", formul
a = y ~ 1, lwd = 2)
```

Finalmente, vamos criar a reta de regressão ótima para os dados com as informações de faturamento, indicando que *y* deve ser estimado a partir de *x*. Para tal, faça:

```
> estetica = aes(x = RENDA, y = FAT)
> base = ggplot(data = tabela, estetica)
> base + geom_point(size = 4) + stat_smooth(method = "lm", formul
a = y ~ x, lwd = 2)
```

Agora que já entendemos como funciona a reta de Regressão Linear na prática, vamos construir um modelo de Regressão Linear a partir dos dados de treinamento para estimar o valor de faturamento nos dados de teste. Para tal, vamos preparar as bases de dados para aplicar a validação cruzada *3-fold*, de forma similar ao que fizemos na prática do capítulo 6:

```
> library("caret")
> idxcv = createFolds(tabela$FAT, k = 3)

> tabTeste1 = tabela[idxcv$Fold1, ]
```

```
> tabTreino1 = tabela[-idxcv$Fold1, ]

> tabTeste2 = tabela[idxcv$Fold2, ]
> tabTreino2 = tabela[-idxcv$Fold2, ]

> tabTreino3 = tabela[-idxcv$Fold3, ]
> tabTeste3 = tabela[idxcv$Fold3, ]
```

Finalmente, chegamos à etapa de modelagem. Vamos ajustar modelos para cada uma das 3 tabelas de treino:

```
> reg1 = lm(FAT ~ RENDA + POP + ESPVIDA +
            TXALFA + NCONC + AREA +
            PRECOM2 + ROUBOSLOJAS +
            ROUBOSRES, data = tabTreino1)

> reg2 = lm(FAT ~ RENDA + POP + ESPVIDA +
            TXALFA + NCONC + AREA +
            PRECOM2 + ROUBOSLOJAS +
            ROUBOSRES, data = tabTreino2)

> reg3 = lm(FAT ~ RENDA + POP + ESPVIDA +
            TXALFA + NCONC + AREA +
            PRECOM2 + ROUBOSLOJAS +
            ROUBOSRES, data = tabTreino3)
```

Em seguida, vamos exibir os principais resultados para cada um dos modelos com o comando *summary*. Por motivo de legibilidade, estamos exibindo apenas os resultados do modelo *reg1*, mas recomendamos que você também execute este comando para *reg2* e *reg3*:

```
> summary(reg1)
##
## Call:
## lm(formula = FAT ~ RENDA + POP + ESPVIDA + TXALFA + NCONC + AR
EA +
##     PRECOM2 + ROUBOSLOJAS + ROUBOSRES, data = tabTreino1)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -33795  -8768  -2891  10320  43605
```

```
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -8.177e+04  1.445e+05  -0.566 0.574302
## RENDA        2.110e+02  2.652e+01   7.956 3.43e-10 ***
## POP          2.099e-01  9.188e-02   2.285 0.026995 *
## ESPVIDA      8.019e+02  2.045e+03   0.392 0.696721
## TXALFA      -1.959e+05  5.866e+04  -3.339 0.001674 **
## NCONC       -5.658e+03  9.225e+02  -6.134 1.83e-07 ***
## AREA         1.162e+04  2.130e+03   5.457 1.87e-06 ***
## PRECOM2     -2.414e+00  2.396e+00  -1.008 0.318948
## ROUBOSLOJAS  6.299e+02  3.667e+02   1.718 0.092557 .
## ROUBOSRES    8.393e+02  2.380e+02   3.526 0.000966 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 16430 on 46 degrees of freedom
## Multiple R-squared:  0.8226, Adjusted R-squared:  0.7879
## F-statistic:  23.7 on 9 and 46 DF,  p-value: 1.927e-14
```

Podemos agora realizar a predição com os dados de teste e calcular os resultados das métricas RMSE e $R_2$, da seguinte forma:

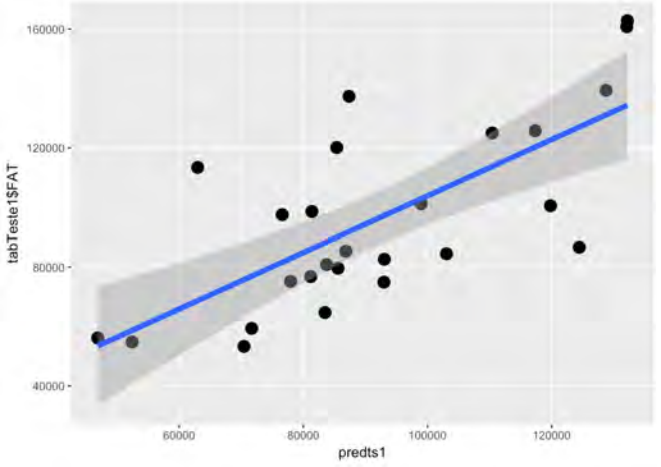
```
> predts1 = as.numeric(predict(reg1, newdata = tabTeste1))
> predts2 = as.numeric(predict(reg2, newdata = tabTeste2))
> predts3 = as.numeric(predict(reg3, newdata = tabTeste3))

> # RMSE
> RMSE1 = sqrt(mean((tabTeste1$FAT - predts1)^2))
> RMSE2 = sqrt(mean((tabTeste2$FAT - predts2)^2))
> RMSE3 = sqrt(mean((tabTeste3$FAT - predts3)^2))
>  # RMSE médio
> RMSERL = (RMSE1 + RMSE2 + RMSE3)/3

> # R2
> R2reg1 = cor(tabTeste1$FAT, predts1)^2
> R2reg2 = cor(tabTeste2$FAT, predts2)^2
> R2reg3 = cor(tabTeste3$FAT, predts3)^2
> # R2 médio
> R2RL = (R2reg1 + R2reg2 + R2reg3)/3
```

Os valores resultantes das métricas RMSE e $R_2$ não serão exibidos neste momento, pois serão analisados no final desta

prática.

Os comandos a seguir possibilitarão a geração de um gráfico mostrando a relação entre os valores reais e os valores preditos:

```
> estetica = aes(y = tabTeste1$FAT,
                 x = predts1)
> ggplot(tabTeste1, estetica) +
  geom_point(size = 4) +
  stat_smooth(method = "lm",
              formula = y ~ x, lwd = 2)
```




Vamos agora aplicar o modelo de Árvore de Regressão para resolver este mesmo problema. Para tal, vamos utilizar a função `ctree` do pacote **party**:

```
> library("party")
```

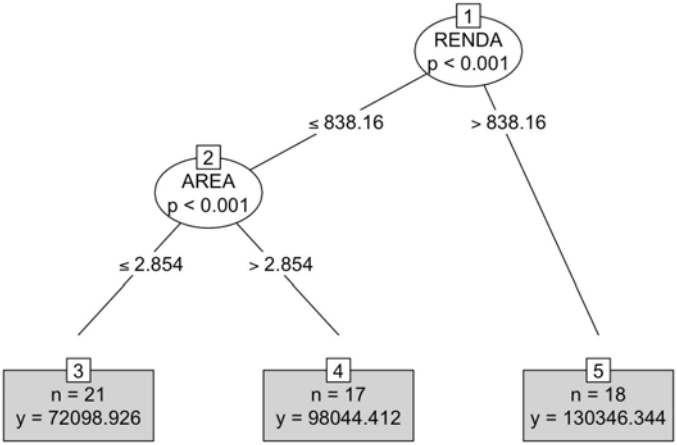
Vamos especificar o critério que queremos utilizar para particionamento da árvore e criar 3 diferentes modelos de Árvore de Regressão para a estimação do faturamento, um com cada tabela de treino. Neste caso, utilizaremos diversos atributos

preditores, e não apenas a renda per capita, como fizemos na Regressão Linear:

```
> controle = ctree_control(mincriterion = 0.95)

> AR1 = ctree(FAT ~ POP + RENDA + ESPVIDA +
              TXALFA + NCONC + AREA + PRECOM2 +
              ROUBOSLOJAS + ROUBOSRES,
            data = tabTreino1, controls = controle)

> AR2 = ctree(FAT ~ POP + RENDA + ESPVIDA +
              TXALFA + NCONC + AREA + PRECOM2 +
              ROUBOSLOJAS + ROUBOSRES,
            data = tabTreino2, controls = controle)

> AR3 = ctree(FAT ~ POP + RENDA + ESPVIDA +
              TXALFA + NCONC + AREA + PRECOM2 +
              ROUBOSLOJAS + ROUBOSRES,
            data = tabTreino3, controls = controle)
```

Para exibir a estrutura e a representação gráfica da árvore gerada, faça:

```
> show(AR1)
##
##   Conditional inference tree with 3 terminal nodes
##
## Response:  FAT
## Inputs:  POP, RENDA, ESPVIDA, TXALFA, NCONC, AREA, PRECOM2, RO
UBOSLOJAS, ROUBOSRES
## Number of observations:  56
##
## 1) RENDA <= 838.1601; criterion = 1, statistic = 19.117
##   2) AREA <= 2.85425; criterion = 1, statistic = 16.915
##     3)*  weights = 21
##   2) AREA > 2.85425
##     4)*  weights = 17
## 1) RENDA > 838.1601
##   5)*  weights = 18

> plot(AR1, terminal_panel =  node_terminal)
```




Agora vamos realizar a predição nas tabelas de teste. Para tal, fazemos:

```
> predAR1 = as.numeric(predict(AR1, newdata = tabTeste1))
> predAR2 = as.numeric(predict(AR2, newdata = tabTeste2))
> predAR3 = as.numeric(predict(AR3, newdata = tabTeste3))
```

Finalmente, para avaliar os modelos usando as métricas RMSE e $R_2$, fazemos:

```
> #  RMSE
> RMSEAR1 = sqrt(mean((tabTeste1$FAT - predAR1)^2))
> RMSEAR2 = sqrt(mean((tabTeste2$FAT - predAR2)^2))
> RMSEAR3 = sqrt(mean((tabTeste3$FAT - predAR3)^2))
> RMSEAR = (RMSEAR1 + RMSEAR2 + RMSEAR3)/3 # RMSE Médio

> # R^2
> R2AR1 = cor(tabTeste1$FAT, predAR1)^2
> R2AR2 = cor(tabTeste2$FAT, predAR2)^2
> R2AR3 = cor(tabTeste3$FAT, predAR3)^2
```

```
> R2AR = (R2AR1 + R2AR2 + R2AR3)/3 # R^2 Médio
```

Os valores resultantes das métricas RMSE e $R_2$ não serão exibidos neste momento, pois serão analisados no final desta prática.

Finalmente, vamos utilizar o modelo KNN para resolver este mesmo problema. Para tal, vamos utilizar a função `knn.reg`, contida no pacote **FNN**:

```
> install.packages("FNN")
> library("FNN")
```

Apenas para exemplificar como funciona este algoritmo, vamos utilizar o valor de 33 vizinhos, armazenado na variável `kvalue` (experimente depois usar outros valores e avalie os resultados). Primeiramente, treinamos um modelo para cada uma das 3 tabelas de treino:

```
> kvalue = 33

> KNN1 = knn.reg(train = tabTreino1[, 2:10],
                 y = tabTreino1[, 11],
                 test = tabTeste1[, 2:10],
                 k = kvalue)

> KNN2 = knn.reg(train = tabTreino2[, 2:10],
                 y = tabTreino2[, 11],
                 test = tabTeste2[, 2:10],
                 k = kvalue)

> KNN3 = knn.reg(train = tabTreino3[, 2:10],
                 y = tabTreino3[, 11],
                 test = tabTeste3[, 2:10],
                 k = kvalue)
```

Em seguida, fazemos a predição para cada uma das tabelas de teste:

```
> predKNN1 = KNN1$pred
> predKNN2 = KNN2$pred
> predKNN3 = KNN3$pred
```

Finalmente, calculamos as métricas de avaliação:

```
> # RMSE
> RMSEKNN1 = sqrt(mean((tabTeste1$FAT - predKNN1)^2))
> RMSEKNN2 = sqrt(mean((tabTeste2$FAT - predKNN2)^2))
> RMSEKNN3 = sqrt(mean((tabTeste3$FAT - predKNN3)^2))
> RMSEKNN = (RMSEKNN1 + RMSEKNN2 + RMSEKNN3)/3 # RMSE médio

> # R^2
> R2KNN1 = cor(tabTeste1$FAT, predKNN1)^2
> R2KNN2 = cor(tabTeste2$FAT, predKNN2)^2
> R2KNN3 = cor(tabTeste3$FAT, predKNN3)^2
> R2KNN = (R2KNN1 + R2KNN2 + R2KNN3)/3 # R2 médio
```

Para concluir a prática, vamos fazer a comparação dos resultados dos 3 algoritmos que usamos para o problema. Vamos comparar os valores resultantes das métricas RMSE e $R_2$, lembrando que, quanto maior o RMSE, melhor é o modelo; e quanto mais R2 estiver próximo de 1, melhor o ajuste do modelo aos dados. O melhor modelo será o mais adequado para realizar a estimação dos dados não rotulados.

Vamos então preparar a exibição de resultados, criando uma tabela para melhor visualização:

```
> tabela_resultados <- c("Modelo", "RMSE", "R2")
> resultadosRL <- c("Regressão Linear", RMSERL, R2RL)
> resultadosAR <- c("Árvore", RMSEAR, R2AR)
> resultadosKNN <- c("KNN", RMSEKNN, R2KNN)
> tabela_resultados <- rbind(tabela_resultados, resultadosRL, res
ultadosAR, resultadosKNN)
> print(tabela_resultados)
##                   [,1]              [,2]
## tabela_resultados "Modelo"          "RMSE"
## resultadosRL      "Regressão Linear" "20986.8978084086"
## resultadosAR      "Árvore"          "32347.8216559812"
## resultadosKNN     "KNN"             "32073.0624494316"
```

```
##                   [,3]
## tabela_resultados "R2"
## resultadosRL      "0.63145531350344"
## resultadosAR      "0.196225465002936"
## resultadosKNN     "0.150895576253887"
```

Como podemos ver, a Regressão Linear foi o melhor modelo dentre os 3 avaliados. Desta forma, ele será o escolhido para a aplicação nos dados não rotulados. Vamos então criar um modelo usando todos os dados, realizar a predição para os dados não rotulados, gravar o resultado da predição em uma nova coluna da tabela e exportar os resultados para uma planilha. Execute o código a seguir, e depois abra o arquivo gerado `faturamentoEstimado.csv` para visualizar os resultados.

```
> # Criação do modelo
> regfinal = lm(FAT ~ RENDA + POP + ESPVIDA +
            TXALFA + NCONC + AREA +
            PRECOM2 + ROUBOSLOJAS +
            ROUBOSRES, data = tabela)

> # Predição de Teste
> predFATNA = predict(regfinal, newdata = tabelaNA)
> predFATNA = as.numeric(predFATNA)
> tabelaNA$FAT_ESTIMADO = predFATNA

> # Exportação dos resultados para planilha
> write.csv2(tabelaNA, "faturamentoEstimado.csv")
```

## 8.2 REGRESSÃO LOGÍSTICA

Para exemplificar a utilização da Regressão Logística no R, vamos utilizar a base de dados **PimaIndiansDiabetes**, do pacote **mlbench**. Este dataset tem o objetivo de determinar a ocorrência de diabetes com base em medidas de diagnóstico, como número de vezes grávida, concentração de glicose plasmática, pressão sanguínea, idade, entre outras. Para executar esta parte prática,

copie os comandos a seguir no seu R Script e execute (ou baixe o script diretamente de https://github.com/tatianaesc/introdatascience/).

Vamos iniciar importando os dados:

```
> library("mlbench")
> data(PimaIndiansDiabetes)
> tabela = as.data.frame(PimaIndiansDiabetes)
```

Em seguida, vamos dividir nossa base de dados aleatoriamente em conjuntos de treino e teste, utilizando uma semente para garantir que você consiga reproduzir os mesmos resultados:

```
> set.seed(7)
> N = nrow(tabela)
> sorteio <- sample(1:N, N*0.75, FALSE)
> baseTreino <- tabela[sorteio, ]
> baseTeste <- tabela[-sorteio, ]
```

Vamos então construir o modelo usando o conjunto de treino e aplicá-lo tanto nos dados de treino quanto nos dados de teste:

```
> # Treina o modelo
> fit <- glm(diabetes~., data=baseTreino, family=binomial(link='l
ogit'))
> print(fit)
##
## Call:  glm(formula = diabetes ~ ., family = binomial(link = "l
ogit"),
##     data = baseTreino)
##
## Coefficients:
## (Intercept)     pregnant      glucose     pressure      tricep
s
##  -8.6606078    0.1119708    0.0414048   -0.0146021   -0.000826
9
##     insulin         mass     pedigree          age
##  -0.0019690    0.0826874    0.6104550    0.0181942
##
## Degrees of Freedom: 575 Total (i.e. Null);  567 Residual
```

```
## Null Deviance:       746.4
## Residual Deviance: 529.1     AIC: 547.1

> # Aplica o modelo nos dados de treino
> probabilitiesTr <- predict(fit, baseTreino[,1:8], type='respons
e')
> predictionsTr <- ifelse(probabilitiesTr > 0.5,'pos','neg')

> # Aplica o modelo nos dados de teste
> probabilitiesTst <- predict(fit, baseTeste[,1:8], type='respons
e')
> predictionsTst <- ifelse(probabilitiesTst > 0.5,'pos','neg')
```

Finalmente, vamos calcular as métricas de classificação e exibir os resultados:

```
> library("caret")

> # Resultados Treino
> resultadoTr <- caret::confusionMatrix(predictionsTr, baseTreino
$diabetes)
> resultadoTr$table # exibe a matriz de confusão
##           Reference
## Prediction neg pos
##        neg 328  87
##        pos  46 115
> resultadoTr$overall[1] # exibe a acurácia
##  Accuracy
## 0.7690972

> # Resultados Teste
> resultadoTst <- caret::confusionMatrix(predictionsTst, baseTest
e$diabetes) # Teste
> resultadoTst$table # exibe a matriz de confusão
##           Reference
## Prediction neg pos
##        neg 108  28
##        pos  18  38
> resultadoTst$overall[1] # exibe a acurácia
##  Accuracy
## 0.7604167
```

## Resumo

Este capítulo apresentou conceitos práticos dos modelos de aprendizagem supervisionada que podem ser usados para resolver problemas de Regressão apresentados no capítulo anterior.

Capítulo 9

# MODELOS DE ASSOCIAÇÃO E AGRUPAMENTO

Conforme apresentamos no capítulo 1, os problemas de Data Science podem ser agrupados de acordo com suas características. Nos capítulos anteriores, apresentamos dois tipos de problemas supervisionados: Classificação e Regressão. Neste capítulo, estudaremos dois tipos de problemas não supervisionados: Associação e Agrupamento.

Estes problemas são ditos não supervisionados porque os objetos de entrada X não possuem rótulos de saída Y associados, ou seja, não há como calcular um erro para verificar se o modelo é adequado. Como não temos disponíveis os rótulos de resposta Y, não estamos interessados em realizar predições, mas sim em descobrir aspectos interessantes sobre os dados. Assim, a aprendizagem não supervisionada engloba técnicas com o objetivo de identificar características chaves para organizar os dados.

A aprendizagem não supervisionada tende a ser mais interessante do que a supervisionada, uma vez que geralmente não se sabe ao certo o que se está procurando e não há uma resposta

certa tal como temos na predição de uma resposta em um problema supervisionado. Muitas vezes, as técnicas não supervisionadas são utilizadas na etapa de análise exploratória de dados para melhor entendimento deles. Além das técnicas de Associação e Agrupamento, existem outras que não serão abordadas neste livro, tais como Redução de Dimensionalidade, Detecção de Outliers e Detecção de Mudanças.

Considerando o esquema básico de um projeto de Data Science, também já apresentado no capítulo 1, assim como nos capítulos 5 a 8, o foco deste capítulo é na etapa de Inferência, ou seja, a escolha do modelo mais adequado para resolver o problema em questão. Relembrando, esta etapa consiste de 3 tarefas:

1. Elencar os modelos possíveis e passíveis para cada tipo de problema;
2. Estimar os parâmetros que compõem os modelos, baseando-se nas instâncias e variáveis pré-processadas;
3. Avaliar os resultados de cada modelo, usando métricas e um processo justo de comparação

## 9.1 PROBLEMAS DE ASSOCIAÇÃO

Para exemplificar os problemas de Associação, considere uma loja virtual com 3 produtos: TV, Tablet e Smartphone. Dado um novo cliente, queremos saber qual produto oferecer: TV, Tablet ou Smartphone. Vamos buscar a resposta via análise do histórico, considerando um grupo de clientes que observaram/adquiriram determinados produtos, ilustrados pela tabela a seguir:

| id | Tablet | TV | Smartphone |
|---|---|---|---|
| A | 1 | 0 | 1 |
| B | 1 | 1 | 0 |
| C | 0 | 0 | 1 |
| D | 0 | 0 | 1 |
| E | 1 | 1 | 0 |
| F | 1 | 0 | 1 |
| G | 1 | 1 | 0 |
| Total | 5 | 3 | 4 |

De acordo com o histórico de compras, parece razoável oferecer ao novo cliente os produtos na seguinte ordem:

- Tablet (5/7 acessos)
- Smartphone (4/7 acessos)
- TV (3/7 acessos)

A próxima pergunta que poderíamos fazer seria: dado que o cliente visualizou o Tablet, qual o próximo item a se oferecer/mostrar? Para tal, podemos observar na tabela anterior todas as transações que acessaram o Tablet: das 5 pessoas que acessaram Tablet, 3 também acessaram TV, e das 5 pessoas que acessaram Tablet, 2 também acessaram Smartphone. Podemos concluir que "se Acesso é Tablet, então Acesso TV", e mostraríamos a TV para este cliente.

No mundo real, podemos citar diversos problemas de Associação que nos deparamos no nosso dia a dia, como a recomendação de produtos no site Submarino.com:

prime FRETE GRÁTIS ILIMITADO
ENTREGA EM 2 DIAS E COM FRETE DE R$0,99? QUERO!
Tablet Samsung Galaxy Tab A SM-P585M 16GB Wi-Fi 4G Tela 10.1" Android Processador Octa-Core - Preto
R$ 1.599,00
Comprar
Quem viu este produto, viu também
R$ 1.249,00
R$ 1.999,00
R$ 1.249,00
R$ 2.459,00
R$ 749,00

Ou ainda, a recomendação de séries e filmes na Netflix:

Formalmente, um problema de Associação pode ser definido como um conjunto de $n$ transações, em que cada transação é composta de um conjunto de *itens*. Cada transação pode ou não conter um conjunto de *atributos*, e o objetivo é construir uma *base de regras* que forme uma *cesta de itens* cujos acessos estão

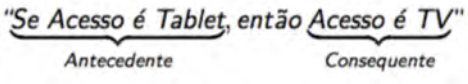
associados. A tabela a seguir ilustra esta definição.

| Transação | Itens | | | Atributos | | |
|---|---|---|---|---|---|---|
| $\mathbf{x}_1$ | 1 | ... | 0 | $x_{11}$ | ... | $x_{1J}$ |
| $\mathbf{x}_2$ | 1 | ... | 1 | $x_{21}$ | ... | $x_{2J}$ |
| $\mathbf{x}_3$ | 0 | ... | 1 | $x_{31}$ | ... | $x_{3J}$ |
| ... | ... | ... | ... | ... | ... | ... |
| $\mathbf{x}_i$ | 0 | ... | 1 | $x_{i1}$ | ... | $x_{iJ}$ |
| ... | ... | ... | ... | ... | ... | ... |
| $\mathbf{x}_n$ | 1 | ... | 1 | $x_{n1}$ | ... | $x_{nJ}$ |

Uma *regra* é uma expressão simbólica representada por:

- Termo antecedente, Premissa ou Lado esquerdo (LHS)
- Termo consequente ou Lado direito (RHS)

Por exemplo:




A parte *antecedente* pode ser formada por uma conjunção, usando o termo linguístico *e*. O *tamanho* da regra indica o número de elementos envolvidos nela. Veja os seguintes exemplos:

- "Se Acesso é Tablet *e* Acesso é TV" (1 conjunção)
- "Se Acesso é Tablet *e* Acesso é TV *e* acesso é Smartphone" (2 conjunções)

Vale a pena observar que a ordem dos fatores na premissa da regra *não* altera o seu resultado final.

Na tabela anterior, o antecedente "Se Acesso é Tablet"

representa:

| id | Tablet |
|---|---|
| A | 1 |
| B | 1 |
| C | 0 |
| D | 0 |
| E | 1 |
| F | 1 |
| G | 1 |
| :-----: | :------: |
| Total | 5 |

Ou de outra forma:

| Item | id | Total |
|---|---|---|
| Tablet | {A, B, E, F, G} | 5 |

Da mesma forma, na tabela anterior, o antecedente "Se Acesso é Tablet e Acesso é TV" representa:

| id | Tablet | TV | Tablet e TV |
|---|---|---|---|
| A | 1 | 0 | 0 |
| B | 1 | 1 | 1 |
| C | 0 | 0 | 0 |
| D | 0 | 0 | 0 |
| E | 1 | 1 | 1 |
| F | 1 | 0 | 0 |
| G | 1 | 1 | 1 |

| | | | |
|---|---|---|---|
| :-----: | :------: | :--: | :----------: |
| Total | 5 | 3 | 3 |

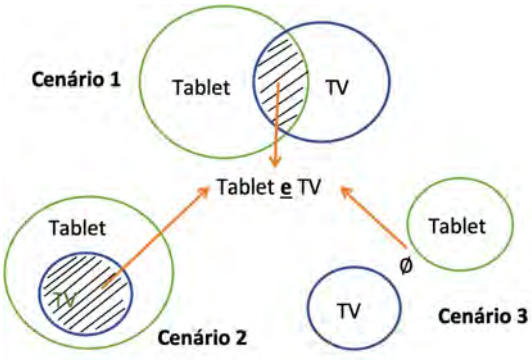
Ou de outra forma:

| Item | id | Total |
|---|---|---|
| Tablet | {A, B, E, F, G} | 5 |
| TV | {B, E, G} | 3 |
| :----------: | :--------------: | :-----: |
| Tablet e TV | {B, E, G} | 3 |

Portanto, a conjunção *e* representa uma intersecção entre os conjuntos (colunas). Assim:

- `#Tablet` = 5
- `#TV` = 3
- `#Tablet e TV` = 3

A operação de intersecção tende sempre a reduzir o número de IDs, como ilustra a figura a seguir.



Há duas métricas relevantes para o processo de mineração (ou descoberta) de regras de associação: *suporte* e *confiança*. Estas métricas possibilitam gerar um conjunto compacto de regras, e que atenda a requisitos mínimos de qualidade. Estas métricas serão detalhadas a seguir.

O *suporte* verifica a frequência de transações que são cobertas pela premissa de regra. Por exemplo, na tabela a seguir:

| id | Tablet | Tablet e TV |
|---|---|---|
| A | 1 | 0 |
| B | 1 | 1 |
| C | 0 | 0 |
| D | 0 | 0 |
| E | 1 | 1 |
| F | 1 | 0 |
| G | 1 | 1 |
| :-------: | :------: | :----------: |
| Total | 5 | 3 |
| Suporte | 0,71 | 0,43 |

temos duas premissas:

$$Prem_1 = \text{“}Se\ acesso\ é\ Tablet\text{”}$$
$$Prem_2 = \text{“}Se\ acesso\ é\ Tablet\ e\ TV\text{”}$$

e o suporte destas premissas é:

$$Sup(Prem_1) = \frac{\#1's}{\#id's} = \frac{5}{7}$$

$$Sup(Prem_2) = \frac{\#1's}{\#id's} = \frac{3}{7}$$

Uma regra de associação X -> Y é considerada *frequente* se o número de vezes em que a união dos conjuntos de itens (X U Y) ocorre em relação ao número total de transações for superior a uma frequência mínima, denominada *suporte mínimo*. Com isso, busca-se identificar que associações surgem em uma quantidade expressiva a ponto de se destacar das demais, e então podem ser tratadas como um padrão.

Já a *confiança* verifica a frequência conjunta de transações que são cobertas tanto pela premissa quanto pelo consequente da regra. No exemplo:

| id | Tablet | Tablet -> TV |
|---|---|---|
| A | 1 | 0 |
| B | 1 | 1 |
| C | 0 | 0 |
| D | 0 | 0 |
| E | 1 | 1 |
| F | 1 | 0 |
| G | 1 | 1 |
| :---------: | :------: | :------------: |
| Total | 5 | 3 |
| Confiança | - | 0,6 |

a regra:

$$R_1 = \text{“}Se\ acesso\ é\ Tablet, então\ acesso\ é\ TV\text{”}$$

tem a seguinte confiança:

$$Conf(R_1) = \frac{Sup(Prem_1 \cap Cons_1)}{Sup(Prem_1)} = \frac{3}{5} = 0{,}6$$

Uma regra de associação X -> Y é considerada *válida* se o número de vezes em que a união dos conjuntos de itens (X U Y) ocorre em relação ao número de vezes em que X ocorre for superior a um valor de confiança mínima. Esta medida busca expressar a qualidade de uma regra: o quanto a ocorrência do antecedente da regra pode assegurar a ocorrência consequente dessa regra.

O algoritmo mais conhecido para problemas de Associação é o *Apriori*, proposto por Agrawal *et al.*, em 1993. Este algoritmo está presente em diversos softwares comerciais e públicos, é de fácil uso e é amplamente utilizado. Existem outros algoritmos para descoberta de associações, como DHP, Partition, DIC, Eclat, MaxEclat, Clique, MaxClique, Cumulate e EstMerge. Todos são inspirados no Apriori, mas estão fora do escopo deste livro.

A ideia básica do Apriori é extrair um conjunto de regras a partir da conjunção de itens mais frequentes na base de dados, até que alguns critérios de parada sejam atendidos, como tamanho máximo da regra, suporte mínimo da premissa e confiança mínima da regra.

Para ilustrar, considere o exemplo até agora usado:

| id | Tablet | TV | Smartphone |
|---|---|---|---|
| A | 1 | 0 | 1 |
| B | 1 | 1 | 0 |
| C | 0 | 0 | 1 |
| D | 0 | 0 | 1 |
| E | 1 | 1 | 0 |
| F | 1 | 0 | 1 |
| G | 1 | 1 | 0 |
| Total | 5 | 3 | 4 |

**Passo 1**: defina os parâmetros usados como base para extração das regras de associação, por exemplo:

- Tamanho máximo da regra = 2
- Suporte mínimo da premissa = 1/7
- Confiança mínima da regra = 2/7

**Passo 2**: gere as primeiras premissas e calcule as suas métricas de suporte:

Premissas:

- $Prem_1$ = "Se Acesso é TV"
- $Prem_2$ = "Se Acesso é Tablet"
- $Prem_3$ = "Se Acesso é Smartphone"

Suporte:

- $Sup(Prem_1) = 3/7$

- $Sup(Prem_2) = 5/7$
- $Sup(Prem_3) = 4/7$

Todas são viáveis, pois atendem ao critério de suporte mínimo da premissa estabelecido anteriormente (1/7).

**Passo 3**: gere as regras e calcule as métricas:

Regras:

- $R_1$ = "Se Acesso é TV, então Acesso é Tablet"
- $R_2$ = "Se Acesso é TV, então Acesso é Smartphone"
- $R_3$ = "Se Acesso é Tablet, então Acesso é TV"
- $R_4$ = "Se Acesso é Tablet, então Acesso é Smartphone"
- $R_5$ = "Se Acesso é Smartphone, então Acesso é TV"
- $R_6$ = "Se Acesso é Smartphone, então Acesso é Tablet"

Confiança:

- $Conf(R_1) = 3/3$
- $Conf(R_2) = 0/3$
- $Conf(R_3) = 3/5$
- $Conf(R_4) = 2/5$
- $Conf(R_5) = 0/4$
- $Conf(R_6) = 2/4$

Podemos cortar as regras $R_2$ e $R_5$, pois não atendem ao critério de confiança mínima da regra estabelecido anteriormente (2/7).

**Passo 4**: refazer os passos 2 e 3 até o critério de parada.

Novas premissas:

- $Prem_4$ = "Se Acesso é TV e Tablet"
- $Prem_5$ = "Se Acesso é TV e Smartphone"
- $Prem_6$ = "Se Acesso é Tablet e TV"
- $Prem_7$ = "Se Acesso é Tablet e Smartphone"
- $Prem_8$ = "Se Acesso é Smartphone e TV"
- $Prem_9$ = "Se Acesso é Smartphone e Tablet"

Suporte:

- $Sup(Prem_4) = 3/7$
- $Sup(Prem_5) = 0/7$
- $Sup(Prem_6) = 3/7$
- $Sup(Prem_7) = 2/7$
- $Sup(Prem_8) = 0/7$
- $Sup(Prem_9) = 2/7$

Podemos cortar as premissas $Prem_5$ e $Prem_8$, pois não atendem ao critério de suporte mínimo da premissa estabelecido anteriormente (1/7).

Novas regras:

- $R_7$ = "Se Acesso é TV e Tablet, então Acesso é Smartphone"
- $R_8$ = "Se Acesso é Tablet e TV, então Acesso é Smartphone"
- $R_9$ = "Se Acesso é Tablet e Smartphone, então Acesso é TV"
- $R_{10}$ = "Se Acesso é Smartphone e Tablet, então Acesso é TV"

Claramente, todas são inviáveis pois a intersecção entre Smartphone e TV é vazia.

**Passo 5**: retornar as regras geradas.

- $R_1$ = "Se Acesso é TV, então Acesso é Tablet"
- $R_3$ = "Se Acesso é Tablet, então Acesso é TV"
- $R_4$ = "Se Acesso é Tablet, então Acesso é Smartphone"
- $R_6$ = "Se Acesso é Smartphone, então Acesso é Tablet"

## 9.2 PROBLEMAS DE AGRUPAMENTO

Problemas de Agrupamento também são conhecidos como problemas de clusterização, e seu objetivo é separar os registros de um conjunto de dados em subconjuntos ou grupos (clusters), de tal forma que elementos em um cluster compartilhem um conjunto de propriedades comuns que os diferencie dos elementos de outros clusters. São os problemas de aprendizagem não supervisionada mais comuns.

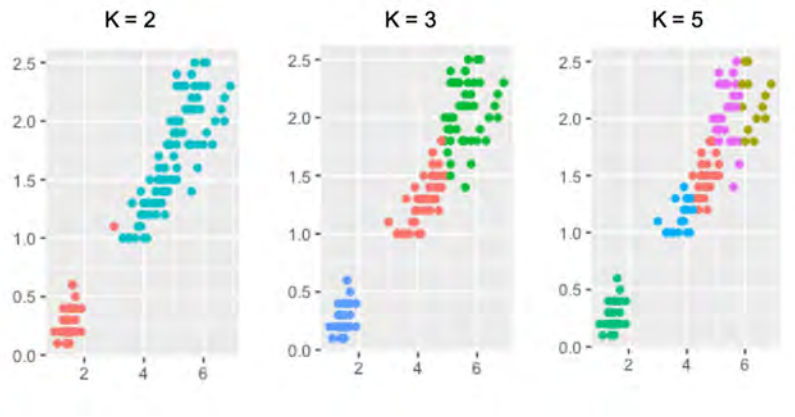
Eles podem ser vistos como um problema de otimização, em que o objetivo é maximizar a similaridade intracluster e minimizar a similaridade intercluster. Isso quer dizer que os grupos são formados de acordo com alguma medida de similaridade: objetos pertencentes a um dado grupo devem ser mais similares entre si (compartilham um conjunto maior de propriedades comuns) do que em relação a objetos pertencentes a outros grupos.

Sendo assim, é necessário definir o que significa que duas observações são similares ou diferentes dentro do domínio do nosso problema. Esta informação geralmente é obtida através da consulta a especialistas do negócio, que auxiliam na interpretação dos resultados.

Alguns algoritmos de Agrupamento requerem que o usuário forneça o número de grupos a formar, e há algoritmos de que

tentam detectar a quantidade de grupos naturais existentes no conjunto de dados de entrada. Geralmente, a presença de dados distribuídos em um espaço de grande dimensionalidade dificulta a detecção de grupos, pois aumenta a complexidade do problema.

Um dos algoritmos de Agrupamento mais conhecidos é o **K-means**, um algoritmo baseado em distâncias. Em linhas gerais, os algoritmos baseados em distância assumem a existência de $n$ pontos de dados $x_1$, $x_2$, ..., $x_n$ tais que cada ponto pertença a um espaço $d$ dimensional $R_d$. Para separá-los em $k$ grupos, deve-se encontrar $k$ pontos $m_j$ em $R_d$, os chamados centroides, de tal forma que a distância entre cada ponto de dado e o centroide mais próximo seja minimizada. O K-means exige que seja informado previamente o parâmetro $k$ (número de grupos) e atribui cada observação a exatamente um dos $k$ grupos. A figura a seguir ilustra os resultados do K-means em uma base de dados considerando 3 diferentes valores de $k$:




Além do K-means, que será detalhado a seguir, são exemplos

de algoritmos baseados em distância: Fuzzy K-Means, K-Modes e K-Medoids. Todos eles também são algoritmos partitivos: inicialmente, escolhem $k$ objetos como sendo os centros dos $k$ grupos. Os objetos são então divididos entre os $k$ grupos de acordo com a medida de similaridade adotada, de modo que cada objeto fique no grupo que forneça o menor valor de distância entre o objeto e o centro do grupo. Em seguida, esses algoritmos utilizam uma estratégia iterativa, que determina se os objetos devem mudar de grupo, fazendo com que cada grupo contenha somente os elementos mais similares entre si.

O algoritmo K-means considera que os registros do conjunto de dados correspondem a pontos no $R_n$, em que cada atributo corresponde a uma dimensão deste espaço. O parâmetro de entrada $k$ determina a quantidade de grupos a serem identificados. Seu funcionamento pode ser resumido nas seguintes etapas:

- Seleciona $k$ pontos do conjunto de dados (sementes), que são os representantes iniciais (centroides), dos $k$ grupos a serem formados.
- Para cada ponto, calcula-se a distância euclidiana deste ponto a cada um dos centroides e atribui-se este ponto ao grupo representado pelo centroide cuja distância é a menor entre todas as calculadas. O resultado desse passo inicial é que cada ponto do conjunto de dados fica associado a um e apenas um dos $k$ grupos.
- Após a alocação inicial, o método segue iterativamente, por meio da atualização dos centroides de cada grupo e da realocação dos pontos ao centroide mais próximo. O novo centroide de cada grupo $G$ é calculado pela média dos pontos alocados a $G$.

- O processo iterativo termina quando os centroides dos grupos param de se modificar, ou após um número preestabelecido de iterações ter sido realizado.

A figura a seguir ilustra a aplicação do K-means em um arquivo com 20 instâncias, utilizando-se k = 3. Inicialmente, a sementes são selecionadas de forma aleatória e cada ponto restante é alocado a algum grupo, em função de sua distância a cada um dos centroides. A seguir, os centroides são atualizados e ocorre nova realocação de pontos. Este processo continua até a convergência.

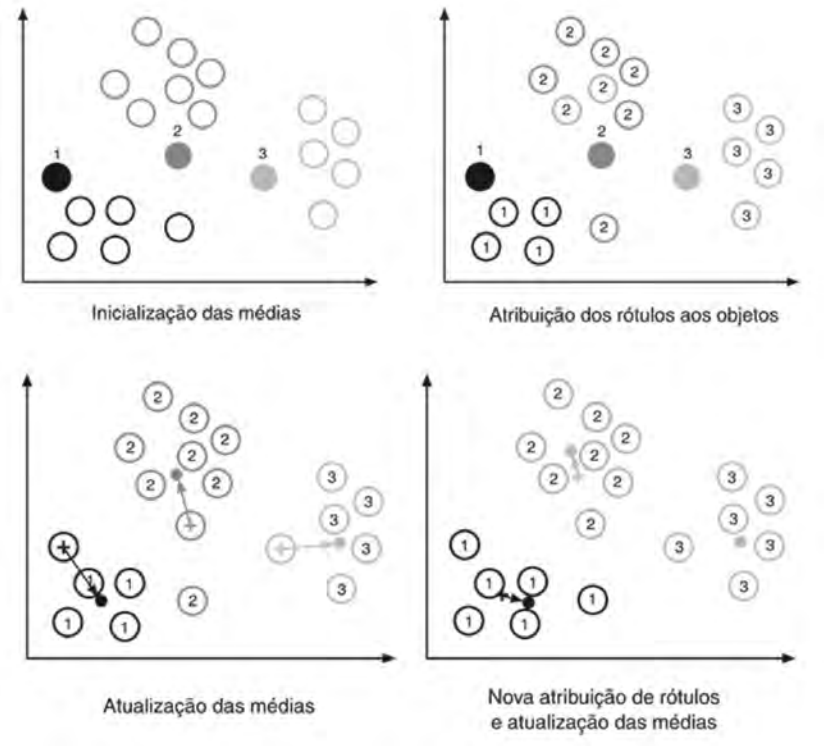



Figura 9.12: Ilustração do K-means (Fonte: GOLDSCHMIDT & PASSOS, 2015)

O K-means apresenta bom desempenho quando os grupos são densos e compactos e bem separados uns dos outros, além de ser rápido e fácil de implementar. Entretanto, há a necessidade de especificar previamente o parâmetro *k* (número de grupos), e para tal, recomendamos que sejam realizados diversos experimentos variando o valor de *k* para determinar o valor adequado.

Este algoritmo não é adequado para descobrir grupos com formas não convexas, grupos de tamanhos muito diferentes e grupos que possuem sobreposição. Adicionalmente, seu desempenho é muito sensível à existência de ruídos no conjunto de dados, visto que pequeno número de dados ruidosos pode influenciar substancialmente os valores médios dos grupos.

## Resumo

Este capítulo apresentou os conceitos teóricos de dois algoritmos aprendizagem não supervisionada para resolver problemas de Associação (Apriori) e Agrupamento (K-means). O próximo capítulo apresentará a parte prática.

## Referências bibliográficas do capítulo:

GOLDSCHMIDT, R.; PASSOS, Emmanuel. *Data mining: um guia prático*. Gulf Professional Publishing, 2005.

Capítulo 10

# PRÁTICAS DE ASSOCIAÇÃO E AGRUPAMENTO

No capítulo anterior, apresentamos os conceitos teóricos de dois algoritmos aprendizagem não supervisionada para resolver problemas de Associação (Apriori) e Agrupamento (K-means). Este capítulo apresenta exemplos práticos destes modelos. O script deste capítulo está disponível em https://github.com/tatianaesc/introdatascience/.

## 10.1 APRIORI

Para exemplificar a utilização do algoritmo Apriori no R, vamos usar agora como dataset uma planilha do Excel similar ao exemplo que foi apresentado no capítulo anterior. Para executar esta parte prática, copie os comandos a seguir no seu R Script e execute (ou baixe o script diretamente de https://github.com/tatianaesc/introdatascience/). Primeiramente, vamos importar os dados.

Importe a tabela a partir do arquivo CSV usando:

```
> tabela = read.table("ExAssocAula.csv", sep = ";", dec = ",", he
ader = T)
```

Como podemos verificar, há 7 transações, cada uma dessas compostas por itens (TV, Tablet, Smartphone) que foram auferidos. O objetivo a partir dessa tabela é extrair regras que associem as transações, de maneira a montar um sistema de recomendação para clientes atuais e novos. Nesse sentido, vamos usar o algoritmo Apriori para extração de regras. Vamos então instalar e referenciar o pacote **arulesViz**:

```
> install.packages("arulesViz")
> library("arulesViz")
```

Em seguida, vamos organizar e transformar os dados na tabela. A primeira transformação a se realizar é remover a coluna `id`, que é inútil para nós:

```
> tabela = tabela[, -1] # remove somente a primeira coluna
> names(tabela)
```

Agora, precisamos transformar os valores inteiros 0 e 1 da tabela no que realmente são: valores lógicos que indicam que a compra do item foi Verdadeira (True) ou Falsa (False). Para tal, faça:

```
> tabela$TV = as.logical(tabela$TV)
> tabela$Tablet = as.logical(tabela$Tablet)
> tabela$Smartphone = as.logical(tabela$Smartphone)
```

Para que possamos trabalhar com a biblioteca *arulesViz* devemos colocar os nossos dados no formato **Transactions**. Ele basicamente transforma uma matriz de incidência ou adjacência (ou uma tabela com valores lógicos – por isso, transformamos os valores 0/1 da tabela em True/False usando `as.logical()`) em uma matriz esparsa, isto é, uma matriz onde somente se guardam

as posições do tipo `TRUE`. Isso tende a reduzir substancialmente a memória alocada para tabelas com um número razoável de transações. Nesse sentido, faça:

```
> tabela = as(tabela, "transactions")
```

Após essa transformação, podemos aplicar o algoritmo Apriori. Em linhas gerais, basta definirmos os parâmetros:

- Tabela com as transações
- Suporte mínimo
- Confiança mínima
- Tamanho mínimo da regra
- Tamanho máximo da regra

No nosso caso, utilizaremos os seguintes valores:

- Tabela com as transações => tabela
- Suporte mínimo => 1/7
- Confiança mínima => 2/7
- Tamanho mínimo da regra => 2
- Tamanho máximo da regra => 3

Para executar o algoritmo Apriori, faça:

```
> param = list(supp = 1/7, conf = 2/7, minlen = 2, maxlen = 3, ex
t = T)
> ruleset = apriori(tabela, parameter = param)
```

Para visualizar os principais resultados, podemos usar a função `summary()` aplicada no objeto `ruleset`:

```
> summary(ruleset)
## set of 4 rules
##
## rule length distribution (lhs + rhs):sizes
## 2
```

```
## 4
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       2       2       2       2       2       2
##
## summary of quality measures:
##     support         confidence     lhs.support          lift

##  Min.   :0.2857   Min.   :0.400   Min.   :0.4286   Min.   :0.7
0
##  1st Qu.:0.2857   1st Qu.:0.475   1st Qu.:0.5357   1st Qu.:0.7
0
##  Median :0.3571   Median :0.550   Median :0.6429   Median :1.0
5
##  Mean   :0.3571   Mean   :0.625   Mean   :0.6071   Mean   :1.0
5
##  3rd Qu.:0.4286   3rd Qu.:0.700   3rd Qu.:0.7143   3rd Qu.:1.4
0
##  Max.   :0.4286   Max.   :1.000   Max.   :0.7143   Max.   :1.4
0
##      count
##  Min.   :2.0
##  1st Qu.:2.0
##  Median :2.5
##  Mean   :2.5
##  3rd Qu.:3.0
##  Max.   :3.0
##
## mining info:
##    data ntransactions   support confidence
##  tabela             7 0.1428571  0.2857143
```

De cima para baixo, obtém-se:

- *A distribuição do tamanho das regras geradas*: no caso, 4 regras de tamanho 2.
- *Estatísticas descritivas a respeito do tamanho das regras*: como somente foram geradas regras de tamanho 2, todas as estatísticas (mínimo, 1º quartil etc.) são iguais.
- *Estatísticas descritivas a respeito das métricas computadas para a base de regras*: observa-se as estatísticas referentes ao

suporte e confiança das regras.

- *Informações da mineração de regras*: base de dados, número de transações, suporte e confiança mínimos.

Após isso, sabe-se que foram geradas no total 4 regras de tamanho 2 (1 premissa e 1 consequente). Para inspecioná-las, basta usar o comando `inspect()`:

```
> inspect(ruleset)
##     lhs             rhs          support   confidence lhs.supp
ort lift
## [1] {Smartphone} => {Tablet}     0.2857143 0.5        0.571428
6   0.7
## [2] {Tablet}     => {Smartphone} 0.2857143 0.4        0.714285
7   0.7
## [3] {TV}         => {Tablet}     0.4285714 1.0        0.428571
4   1.4
## [4] {Tablet}     => {TV}         0.4285714 0.6        0.714285
7   1.4
##     count
## [1] 2
## [2] 2
## [3] 3
## [4] 3
```

Podemos verificar que foram geradas as mesmas regras extraídas no exemplo teórico do capítulo anterior. Uma das mais relevantes é a regra TV => Tablet, dado seu valor de suporte e confiança serem bastante elevados. Logo, se um cliente tiver comprado uma TV, uma boa recomendação seria oferecer a esse cliente um Tablet.

Para se organizar as regras, pode-se escolher uma das métricas como forma de ordenação. Usualmente, opta-se por ordenar as regras usando primeiro a confiança, em seguida, o suporte da regra e, depois, o suporte da premissa. Para realizar isso, faça:

```
> ruleset = sort(ruleset, by = c("confidence", "support", "lhs.su
```

```
pport"))
> inspect(ruleset)
##     lhs             rhs          support   confidence lhs.supp
ort lift
## [1] {TV}         => {Tablet}     0.4285714 1.0        0.428571
4   1.4
## [2] {Tablet}     => {TV}         0.4285714 0.6        0.714285
7   1.4
## [3] {Smartphone} => {Tablet}     0.2857143 0.5        0.571428
6   0.7
## [4] {Tablet}     => {Smartphone} 0.2857143 0.4        0.714285
7   0.7
##     count
## [1] 3
## [2] 3
## [3] 2
## [4] 2
```

Agora, vamos demonstrar outro exemplo de aplicação do Apriori, utilizando um caso mais próximo do mundo real, referente a uma empresa calçadista. Vamos usar como dataset outra planilha, disponível em https://github.com/tatianaesc/introdatascience/.

Neste exemplo, o principal objetivo é construir um sistema de recomendação de recompra para clientes nela cadastrado. Para tanto, foi levantado o volume de compras de cada cliente (transações) em um determinado período (aproximadamente um mês) para diversos modelos de calçados (itens). Será necessário realizarmos algumas etapas de pré-processamento de dados antes de aplicar o algoritmo Apriori.

O primeiro passo é importar o arquivo `ExRealAssoc.csv` :

```
> tabela = read.table("ExRealAssoc.csv", sep = ";", dec = ",", he
ader = T)
```

Para cada cliente temos o segmento que participa, bem como o

quanto foi adquirido de cada modelo de calçado. Pode-se verificar qual segmento é mais frequente, fazendo:

```
> sort(table(tabela$Segmento), decreasing = T)
##
##                   SAPATARIA                     SURFSHOP
##                        2250                         1013
##            BOUTIQUE UNISSEX       LOJAS DE DEPARTAMENTO
##                         681                          591
## LOJAS DE MATERIAL ESPORTIVO        BOUTIQUE DE CALCADOS
##                         418                          135
##          BOUTIQUE MASCULINA           BOUTIQUE DE TENIS
##                          20                           14
##          SAPATARIA INFANTIL           BOUTIQUE FEMININA
##                          10                            8
##           BOUTIQUE INFANTIL            LOJAS MODA PRAIA
##                           7                            4
##              LOJAS VIRTUAIS
##                           3
```

Agora, precisamos categorizar as vendas de cada produto por níveis ou tipos lógicos quaisquer. Por exemplo, poderíamos trabalhar com uma das três ideias:

- Se houve venda daquele produto para um cliente coloca-se 1, caso contrário, é zero (por exemplo, transformar a variável JY em um tipo binário fazendo JY>0).
- Trabalhar com faixas de vendas predeterminadas por um especialista, categorizando cada produto, por exemplo, em vendas baixas, médias e altas para um determinado cliente.
- Trabalhar com faixas, mas determinada pelos dados, usando algum critério como: dividir em regiões de tamanho igual ao domínio de valores de cada item; encontrar regiões com a mesma frequência de clientes, isto é, proporção igual de clientes em cada faixa etc.

Para esse caso, como não se conta com um especialista, vamos

trabalhar com faixas criadas a partir dos dados. Para criar essas faixas facilmente, basta usar a função `discretize()` com alguns argumentos definidos pelo usuário:

- `method` : "interval" para regiões iguais e "frequency" para proporção igual para cada região;
- `breaks` : número de categorias/faixas definidas pelo usuário;
- `labels` : nome dado para cada faixa.

Aplicando isso para o item NK, tem-se:

```
> ClientesCat = discretize(tabela$NK, method="interval", breaks =
3, labels = c("Baixo", "Médio", "Alto"))

# mostra a quantidade de elementos em cada categoria
> table(ClientesCat)
## ClientesCat
## Baixo Médio  Alto
##  5143     7     4

# mostra os valores das faixas
> discretize(tabela$NK, method="interval", breaks = 3, onlycuts=T
)
## [1]     0 10744 21488 32232
```

Portanto, basta repetir essa função para os demais itens:

```
# Legenda: VB = Volume Baixo, VM = Volume Médio e VA = Volume Alt

> CatFLO = discretize(tabela$FLO, method="interval", breaks = 3,
labels = c("VB", "VM", "VA"))
> CatJY = discretize(tabela$JY, method="interval", breaks = 3, la
bels = c("VB", "VM", "VA"))
> CatKIV = discretize(tabela$KIV, method="interval", breaks = 3,
labels = c("VB", "VM", "VA"))
> CatLEV = discretize(tabela$LEV, method="interval", breaks = 3,
labels = c("VB", "VM", "VA"))
> CatLIP = discretize(tabela$LIP, method="interval", breaks = 3,
labels = c("VB", "VM", "VA"))
```

```
> CatNAR = discretize(tabela$NAR, method="interval", breaks = 3,
labels = c("VB", "VM", "VA"))
> CatNK = discretize(tabela$NK, method="interval", breaks = 3, la
bels = c("VB", "VM", "VA"))
> CatSPD = discretize(tabela$SPD, method="interval", breaks = 3,
labels = c("VB", "VM", "VA"))
> CatSUM = discretize(tabela$SUM, method="interval", breaks = 3,
labels = c("VB", "VM", "VA"))
> CatTRP = discretize(tabela$TRP, method="interval", breaks = 3,
labels = c("VB", "VM", "VA"))
```

Após criar todas essas categorias na tabela, devemos agora criar um data frame contendo todas essas informações, assim como o segmento de cada cliente:

```
> Seg = tabela$Segmento # separa a variável Segmento
> tabitens = data.frame(Seg, CatFLO, CatJY, CatKIV, CatLEV, CatLI
P, CatNAR, CatNK, CatSPD, CatSUM, CatTRP)
```

Depois de toda essa organização, basta transformar o tipo de *tabitens*: do tipo `data frame` para o tipo `transactions`:

```
> tabitens = as(tabitens, "transactions")
```

Finalmente, podemos extrair e visualizar as regras de associação:

```
> param = list(supp = 0.5, conf = 0.9, minlen = 2, maxlen = 5, ex
t = TRUE)
> ruleset = apriori(tabitens, parameter = param)
> summary(ruleset)
## set of 2550 rules
##
## rule length distribution (lhs + rhs):sizes
##    2    3    4    5
##   90  360  840 1260
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.000   4.000   4.000   4.282   5.000   5.000
##
## summary of quality measures:
##     support         confidence      lhs.support          lift
```

```
##  Min.   :0.9940   Min.   :0.9973   Min.   :0.9944   Min.   :1.
000
##  1st Qu.:0.9957   1st Qu.:0.9992   1st Qu.:0.9961   1st Qu.:1.
000
##  Median :0.9963   Median :0.9996   Median :0.9967   Median :1.
001
##  Mean   :0.9962   Mean   :0.9995   Mean   :0.9967   Mean   :1.
001
##  3rd Qu.:0.9967   3rd Qu.:1.0000   3rd Qu.:0.9973   3rd Qu.:1.
002
##  Max.   :0.9988   Max.   :1.0000   Max.   :0.9994   Max.   :1.
002
##      count
##  Min.   :5123
##  1st Qu.:5132
##  Median :5135
##  Mean   :5134
##  3rd Qu.:5137
##  Max.   :5148
##
## mining info:
##      data ntransactions support confidence
##  tabitens          5154     0.5        0.9
```

Como foram geradas muitas regras (2550), podemos tentar aumentar o valor do suporte e da confiança, e reduzir o valor do tamanho máximo para diminuir o número de regras:

```
> param = list(supp = 0.9, conf = 1, minlen = 2, maxlen = 3, ext
= TRUE)
> ruleset = apriori(tabitens, parameter = param)
> summary(ruleset)
## set of 74 rules
##
## rule length distribution (lhs + rhs):sizes
##  2  3
##  8 66
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.000   3.000   3.000   2.892   3.000   3.000
##
## summary of quality measures:
```

```
##     support         confidence  lhs.support          lift
##  Min.   :0.9959   Min.   :1    Min.   :0.9959   Min.   :1.001
##  1st Qu.:0.9969   1st Qu.:1    1st Qu.:0.9969   1st Qu.:1.001
##  Median :0.9973   Median :1    Median :0.9973   Median :1.001
##  Mean   :0.9973   Mean   :1    Mean   :0.9973   Mean   :1.001
##  3rd Qu.:0.9977   3rd Qu.:1    3rd Qu.:0.9977   3rd Qu.:1.001
##  Max.   :0.9981   Max.   :1    Max.   :0.9981   Max.   :1.002
##      count
##  Min.   :5133
##  1st Qu.:5138
##  Median :5140
##  Mean   :5140
##  3rd Qu.:5142
##  Max.   :5144
##
## mining info:
##      data ntransactions support confidence
##  tabitens          5154     0.9          1
```

Vamos então ordenar as 74 regras resultantes por confiança, suporte da regra e premissa, e então inspecioná-las:

```
> ruleset = sort(ruleset, by = c("confidence", "support", "lhs.su
pport"))
> inspect(ruleset)
##      lhs                       rhs         support   confidence
 lhs.support
## [1]  {CatFLO=VB}           => {CatLEV=VB} 0.9980598 1
 0.9980598
## [2]  {CatFLO=VB}           => {CatJY=VB}  0.9980598 1
 0.9980598
## [3]  {CatFLO=VB}           => {CatNAR=VB} 0.9980598 1
 0.9980598
## [4]  {CatFLO=VB,CatLEV=VB} => {CatJY=VB}  0.9980598 1
 0.9980598
## [5]  {CatFLO=VB,CatJY=VB}  => {CatLEV=VB} 0.9980598 1
 0.9980598
```

```
## [6]  {CatFLO=VB,CatLEV=VB} => {CatNAR=VB} 0.9980598 1
 0.9980598
## [7]  {CatFLO=VB,CatNAR=VB} => {CatLEV=VB} 0.9980598 1
 0.9980598
## [8]  {CatFLO=VB,CatJY=VB}  => {CatNAR=VB} 0.9980598 1
 0.9980598
## [9]  {CatFLO=VB,CatNAR=VB} => {CatJY=VB}  0.9980598 1
 0.9980598
## [10] {CatLEV=VB,CatTRP=VB} => {CatNAR=VB} 0.9980598 1
 0.9980598
## (parte dos resultados omitidos)
## [70] 1.000971 5136
## [71] 1.000582 5136
## [72] 1.000582 5136
## [73] 1.000971 5133
## [74] 1.000582 5133
```

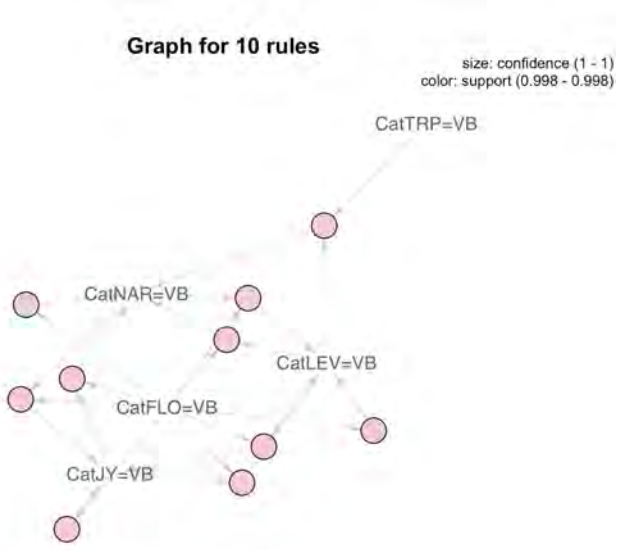
Também podemos inspecionar apenas as 10 regras criadas com maior suporte/confiança:

```
> inspect(head(sort(ruleset),10))
##      lhs                       rhs         support   confidence
 lhs.support
## [1]  {CatFLO=VB}           => {CatLEV=VB} 0.9980598 1
 0.9980598
## [2]  {CatFLO=VB}           => {CatJY=VB}  0.9980598 1
 0.9980598
## [3]  {CatFLO=VB}           => {CatNAR=VB} 0.9980598 1
 0.9980598
## [4]  {CatFLO=VB,CatLEV=VB} => {CatJY=VB}  0.9980598 1
 0.9980598
## [5]  {CatFLO=VB,CatJY=VB}  => {CatLEV=VB} 0.9980598 1
 0.9980598
## [6]  {CatFLO=VB,CatLEV=VB} => {CatNAR=VB} 0.9980598 1
 0.9980598
## [7]  {CatFLO=VB,CatNAR=VB} => {CatLEV=VB} 0.9980598 1
 0.9980598
## [8]  {CatFLO=VB,CatJY=VB}  => {CatNAR=VB} 0.9980598 1
 0.9980598
## [9]  {CatFLO=VB,CatNAR=VB} => {CatJY=VB}  0.9980598 1
 0.9980598
## [10] {CatLEV=VB,CatTRP=VB} => {CatNAR=VB} 0.9980598 1
 0.9980598
##      lift     count
```

```
## [1]  1.000971 5144
## [2]  1.000582 5144
## [3]  1.000582 5144
## [4]  1.000582 5144
## [5]  1.000971 5144
## [6]  1.000582 5144
## [7]  1.000971 5144
## [8]  1.000582 5144
## [9]  1.000582 5144
## [10] 1.000582 5144
```

Podemos plotar estas 10 regras em um gráfico, fazendo:

```
> subruleset = head(sort(ruleset, by="confidence"), 10)
> plot(subruleset, method="graph", measure=c("confidence"), shadi
ng="support")
```



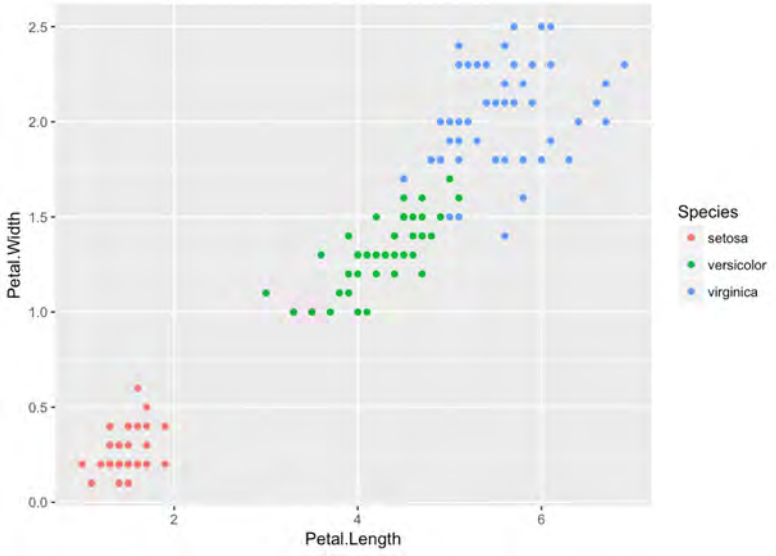

## 10.2 K-MEANS

Para exemplificar a utilização do algoritmo K-means no R, vamos usar dois diferentes datasets nativos do R: Iris e Cars. Para

executar esta parte prática, copie os comandos a seguir no seu R Script e execute (ou baixe o script diretamente de https://github.com/tatianaesc/introdatascience/).

O dataset Iris contém dados sobre comprimento da sépala, largura da sépala, comprimento da pétala e largura da pétala de flores de diferentes espécies. Após alguma exploração, podemos ver que `Petal.Length` e `Petal.Width` são semelhantes entre as mesmas espécies, mas variam consideravelmente entre diferentes espécies:

```
> library(datasets)
> library(ggplot2)
> ggplot(iris, aes(Petal.Length, Petal.Width, color = Species)) +
 geom_point()
```




Agora que examinamos os dados, vamos tentar agrupá-los. Como as atribuições iniciais do cluster são aleatórias, vamos

definir uma semente para garantir a reprodutibilidade.

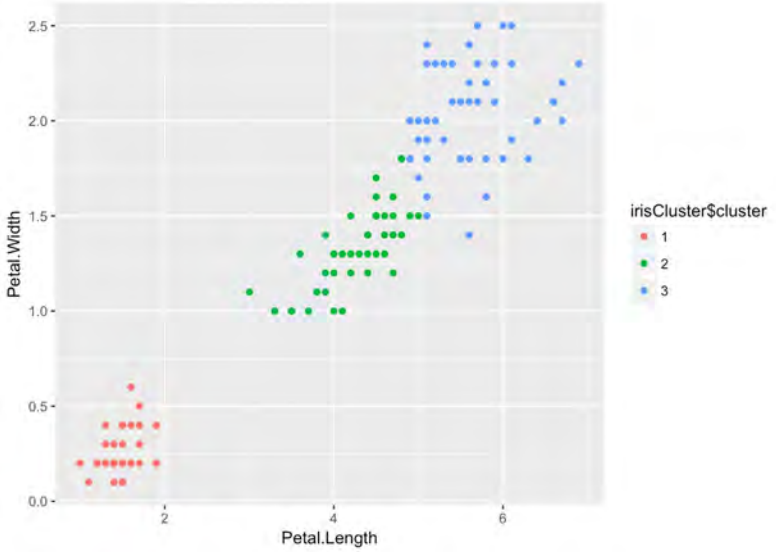
```
> set.seed(20)
> irisCluster <- kmeans(iris[, 3:4], 3, nstart = 20)
> irisCluster
## K-means clustering with 3 clusters of sizes 50, 52, 48
##
## Cluster means:
##   Petal.Length Petal.Width
## 1     1.462000    0.246000
## 2     4.269231    1.342308
## 3     5.595833    2.037500
##
## Clustering vector:
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [71] 2 2 2 2 2 2 2 3 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3
## [106] 3 2 3 3 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 2 3
## [141] 3 3 3 3 3 3 3 3 3
##
## Within cluster sum of squares by cluster:
## [1]  2.02200 13.05769 16.29167
##  (between_SS / total_SS =  94.3 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"

## [5] "tot.withinss" "betweenss"    "size"         "iter"

## [9] "ifault"
```

Em seguida, vamos comparar os clusters com as classes reais.

```
> table(irisCluster$cluster, iris$Species)
##
##     setosa versicolor virginica
##   1     50          0         0
##   2      0         48         4
##   3      0          2        46
```

Como podemos ver, os dados pertencentes às espécies `setosa` foram agrupados no cluster 3, `versicolor` no cluster 2, e `virginica` no cluster 1. O algoritmo classificou erroneamente dois pontos de dados pertencentes ao `versicolor` e seis pontos de dados pertencentes a `virginica`. Também podemos plotar os dados para visualizar os clusters:

```
> irisCluster$cluster <- as.factor(irisCluster$cluster)
> ggplot(iris, aes(Petal.Length, Petal.Width, color = irisCluster
$cluster)) + geom_point()
```




Vamos agora limpar a `console` e o `global environment` no R para trabalhar com um conjunto de dados diferente. Faça:

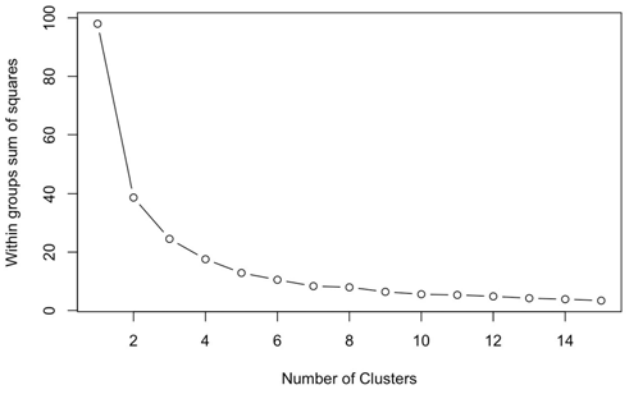
```
> # Limpa a console
> cat("\014")
> # Limpa o global environment
> rm(list = ls())
> # Carrega um novo conjunto de dados
> mydata = cars
```

Vamos preparar os dados, eliminando os `missings` e padronizando as variáveis.

```
> mydata <- na.omit(mydata) # eliminação de missings
> mydata <- scale(mydata) # padronização de variáveis
```

Em seguida, vamos determinar o número de clusters usando um `loop` for, e plotar este resultado em um gráfico:

```
> wss <- (nrow(mydata)-1)*sum(apply(mydata,2,var))
> for (i in 2:15) wss[i] <- sum(kmeans(mydata,
                                       centers=i)$withinss)
> plot(1:15, wss, type="b", xlab="Number of Clusters",
     ylab="Within groups sum of squares")
```

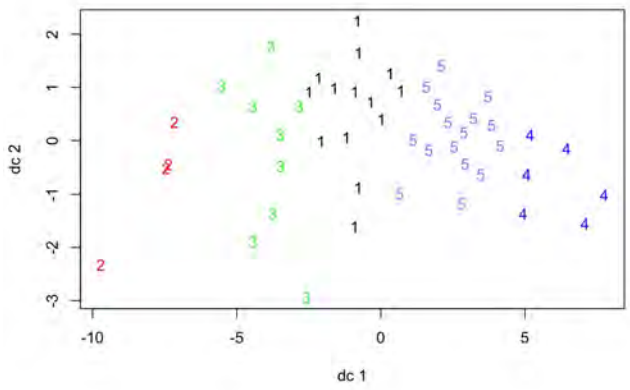



A partir destes resultados, vamos escolher usar k = 5. Aplicamos o K-means nos dados:

```
> fit <- kmeans(mydata, 5)
```

Em seguida, podemos plotar os clusters, usando a função `plotcluster` do pacote `fpc`, que precisará ser instalado e carregado:

```
> install.packages("fpc")
> library(fpc)
> plotcluster(mydata, fit$cluster)
```




Também podemos calcular as médias de cada um dos clusters:

```
> aggregate(mydata,by=list(fit$cluster),FUN=mean)
##   Group.1      speed        dist
## 1       1  0.4106609  0.05066922
## 2       2  1.6737131  2.11568945
## 3       3  0.8279259  0.99679034
## 4       4 -1.6831692 -1.25394311
## 5       5 -0.5762602 -0.62468125
`
```

Finalmente, vamos adicionar e exibir a atribuição de clusters no dataset, fazendo:

```
> mydata <- data.frame(mydata, fit$cluster)
> mydata
##          speed        dist fit.cluster
## 1  -2.15596948 -1.59025960           4
## 2  -2.15596948 -1.27981361           4
## 3  -1.58860909 -1.51264810           4
## 4  -1.58860909 -0.81414462           4
```

```
## 5  -1.39948896 -1.04697911          4
## 6  -1.21036883 -1.27981361          4
## 7  -1.02124870 -0.96936762          5
## 8  -1.02124870 -0.65892162          5
## 9  -1.02124870 -0.34847563          5
## 10 -0.83212857 -1.00817336          5
(...)
## 45  1.43731298  0.42763936          3
## 46  1.62643311  1.04853134          3
## 47  1.62643311  1.90225783          2
## 48  1.62643311  1.94106357          2
## 49  1.62643311  2.98881880          2
## 50  1.81555324  1.63061758          2
```

## Resumo

Este capítulo apresentou conceitos práticos dos modelos de aprendizagem não supervisionada apresentados no capítulo anterior.

Capítulo 11

# CONCLUSÃO

Ao longo de todo o livro, utilizamos como guia o esquema básico de um projeto de Data Science, que pode ser resumido em 7 etapas:

1. Problema, necessidade ou ideia;
2. Coleta e análise de dados;
3. Pré-processamento;
4. Modelagem e inferência;
5. Pós-processamento;
6. Apresentação de resultados
7. Implantação do modelo e geração de valor.

Conforme exposto no capítulo 1, o objetivo principal deste livro foi mostrar como utilizar a Ciência de Dados para resolver problemas e gerar produtos que agreguem valor ao negócio, aprendendo com os dados. Assim, focamos no processo e nas técnicas relacionadas aos algoritmos preditivos mais comumente utilizados em Data Science, mas mostrando também a importância da etapa de preparação dos dados brutos, limpeza e análise exploratória.

Nosso foco principal foi nas etapas 3 e 4 do esquema básico do projeto de Data Science: os capítulos 1 e 3 apresentaram,

respectivamente, conceitos introdutórios sobre Ciência de Dados, linguagem R, Estatística e Álgebra Linear, com o objetivo de fornecer os conceitos teóricos necessários para os capítulos seguintes. O capítulo 4, por sua vez, abordou a etapa 3 do esquema básico do projeto de Data Science, e os capítulos de 5 a 10 abordaram a etapa 4.

## 11.1 SUGESTÃO DE TEMPLATE BÁSICO PARA PROJETO DE CIÊNCIA DE DADOS

Para a criação de um projeto prático completo de Ciência de Dados, é interessante utilizarmos uma estrutura inspirada nas 7 etapas do esquema básico. O código a seguir ilustra uma sugestão de template de uma estrutura geral de um projeto de Ciência de Dados, de ponta a ponta, assumindo que grande parte das duas primeiras etapas (entendimento do problema e coleta de dados necessários) já foram previamente realizadas junto aos especialistas, e que as duas últimas etapas (apresentação dos resultados e implantação do modelo em produção) serão realizadas posteriormente. Este template também está disponível para download em https://github.com/tatianaesc/introdatascience/.

Sugerimos que você o utilize como base para os seus projetos, preenchendo cada uma das etapas com os códigos correspondentes, lembrando que nem sempre o fluxo é seguido nesta ordem e de forma linear. Por exemplo, você poderá descobrir que é necessário realizar algum tipo de transformação de dados apenas no momento de avaliar os modelos construídos; ou ainda, poderá sentir necessidade de realizar análises exploratórias adicionais dos dados somente no momento de realizar o ajuste dos

hiperparâmetros dos modelos, sendo necessário, assim, retornar a etapas anteriores. À medida que você avançar nos seus estudos, você poderá adaptar e evoluir este template conforme suas necessidades.

```
# 1. Preparação do problema
# a) Carga de pacotes
# b) Carga de dataset
# c) Divisão do dataset em conjunto de treino e teste (validação
final)

# 2. Análise exploratória de dados
# a) Estatísticas Descritivas
# b) Visualizações de Dados

# 3. Pré-processamento
# a) Limpeza de Dados
# b) Seleção de atributos
# c) Transformação de Dados

# 4. Modelagem e Inferência
# a) Escolha de procedimentos e métricas de avaliação
# b) Criação de Algoritmos
# c) Comparação de Algoritmos
# d) Melhoria do desempenho dos modelos através de ajuste de hipe
rparâmetros

# 5. Pós-processamento
# a) Escolha e construção do modelo final com todo o conjunto de
treinamento
# b) Predições no conjunto de teste (validação final)
# c) Salvamento do modelo para uso posterior
```

O primeiro bloco, **Preparação do problema**, consiste em realizar a carga de todos os recursos necessários para trabalhar no problema, incluindo os pacotes R e o dataset utilizado. Também é realizada a separação em conjuntos de treino e teste: caso você tenha um número de dados suficientemente grande, sugere-se que seja primeiro realizada uma divisão do tipo hold-out (separando, por exemplo, 90% dos dados para treinamento e 10% dos dados

para teste do modelo final). Em seguida, deve-se utilizar a validação cruzada 10-fold neste conjunto de treinamento para a avaliação dos modelos (criando, neste processo, 10 diferentes configurações com 9 partições de treino e 1 de teste). Caso você não tenha um número de dados suficientemente grande, você pode optar apenas pela validação cruzada ou pela divisão *hold-out* simples (treino e teste).

Já o bloco **Análise exploratória de dados** tem o objetivo de entender e explorar os dados disponíveis, o que inclui a geração de estatísticas descritivas e visualizações gráficas dos dados; e o bloco **Pré-processamento** inclui tarefas de preparação dos dados como limpeza de dados (como remoção de duplicatas, tratamento de valores faltantes), seleção de atributos (removendo, por exemplo, atributos redundantes ou pouco relevantes) e transformação de dados (por exemplo, normalização e padronização de dados).

O bloco **Modelagem e Inferência**, por sua vez, consiste em elencar e testar algoritmos adequados para os dados (idealmente, utilizando validação cruzada) e o problema que se quer resolver, incluindo tarefas de definição de procedimentos de avaliação (por exemplo, validação cruzada) e métricas de avaliação (por exemplo, acurácia ou RMSE). Nessa etapa, os resultados dos algoritmos são comparados e aqueles com resultados mais promissores podem ter seus hiperparâmetros ajustados, de forma a encontrar sua melhor configuração. Durante essa etapa, você também pode combinar diversos modelos de Machine Learning em um *ensemble*, mas este assunto está fora do escopo deste livro.

Finalmente, o bloco **Pós-processamento** consiste em finalizar o modelo que foi mais bem avaliado no bloco anterior, o que

envolve tarefas como o retreinamento do modelo utilizando todo o conjunto de treinamento (o mesmo que foi utilizado para a aplicação da validação cruzada, na etapa de avaliação dos modelos), teste final do modelo (utilizando o conjunto de teste separado no primeiro bloco para simular dados não-vistos) e o salvamento do modelo em arquivo para uso posterior em dados não vistos.

## 11.2 EXEMPLO DE UM PROJETO COMPLETO USANDO O TEMPLATE

Para ilustrar a utilização do template, vamos trabalhar em um problema de Classificação binária usando o **dataset de câncer de mama de Wisconsin**, já utilizado em capítulos anteriores, onde cada exemplo do dataset representa uma amostra de tecido de câncer de mama. Você poderá fazer download deste código completo em https://github.com/tatianaesc/introdatascience/. Caso seja necessário instalar algum dos pacotes mencionados neste script, basta digitar o comando:

```
install.packages("nome_do_pacote")
```

No bloco de **preparação do problema**, vamos carregar os pacotes necessários e o dataset. Também vamos dividir o dataset em 80% para treinamento e 20% para teste (que serão guardados para a etapa de validação final do modelo):

```
# 1. Preparação do problema
set.seed(7) # definição de semente

# a) Carga de pacotes
library(mlbench)
library(caret)
```

```
# b) Carga de dataset
data(BreastCancer)

# c) Divisão do dataset em conjunto de treino e teste (validação
final)
# separação do dataset em 80% para treinamento (usado para treina
r e testar os modelos)
# e 20% para teste (validação final) do modelo
indiceTeste <- createDataPartition(BreastCancer$Class, p=0.80, li
st=FALSE)
conjTeste <- BreastCancer[-indiceTeste,] # conjunto de teste
dataset <- BreastCancer[indiceTeste,] # conjunto de treinamento
```

A seguir, entramos no bloco de **análise exploratória de dados** e iniciamos exibindo algumas estatísticas descritivas dos dados, como dimensões do dataset, suas primeiras linhas e os tipos dos atributos:

```
# 2. Análise exploratória de dados

# a) Estatísticas Descritivas
dim(dataset) # dimensões do dataset
head(dataset, n=10) # 10 primeiras linhas
sapply(dataset, class) # tipos dos atributos
```

```
> dim(dataset) # dimensões do dataset
[1] 560  11
> head(dataset, n=10) # 10 primeiras linhas
        Id Cl.thickness Cell.size Cell.shape Marg.adhesion Epith.c.size
2  1002945            5         4          4             5            7
4  1016277            6         8          8             1            3
5  1017023            4         1          1             3            2
6  1017122            8        10         10             8            7
7  1018099            1         1          1             1            2
9  1033078            2         1          1             1            2
10 1033078            4         2          1             1            2
11 1035283            1         1          1             1            1
12 1036172            2         1          1             1            2
14 1043999            1         1          1             1            2

2  Bare.nuclei Bl.cromatin Normal.nucleoli Mitoses     Class
4           10           3               2       1    benign
5            4           3               7       1    benign
6            1           3               1       1    benign
7           10           9               7       1 malignant
9           10           3               1       1    benign
10           1           1               1       5    benign
11           1           2               1       1    benign
12           1           3               1       1    benign
14           1           2               1       1    benign
```

Figura 11.1: Resultados

```
> sapply(dataset, class) # tipos dos atributos
$Id
[1] "character"

$Cl.thickness
[1] "ordered" "factor"

$Cell.size
[1] "ordered" "factor"

$Cell.shape
[1] "ordered" "factor"

$Marg.adhesion
[1] "ordered" "factor"

$Epith.c.size
[1] "ordered" "factor"

$Bare.nuclei
[1] "factor"

$Bl.cromatin
[1] "factor"

$Normal.nucleoli
[1] "factor"

$Mitoses
[1] "factor"

$Class
[1] "factor"
```

Figura 11.2: Resultados

A execução deste trecho de código indica que o atributo *Id* provavelmente não será útil, então poderemos eliminá-lo. Além disso, fora o atributo *Id*, os demais são do tipo *factors*, porém, dado que existe uma relação ordinal entre os valores dos atributos, podemos convertê-los para valores numéricos. Já estamos adiantando tarefas do bloco seguinte, de pré-processamento de dados. Isso mostra que você não necessariamente precisa utilizar o template estritamente na ordem que sugerimos, sendo apropriado

ir realizando as operações na ordem que fizer sentido para cada problema.

```
# Remoção do atributo Id
dataset <- dataset[,-1]

# Conversão dos valores de entrada para numéricos
for(i in 1:9) {
  dataset[,i] <- as.numeric(as.character(dataset[,i]))
}
```

Continuamos a parte de **estatísticas descritivas** exibindo um resumo estatístico e a distribuição de classes. Verificaremos a presença de alguns NA's no atributo *Bare.nuclei*, podendo ser necessário o tratamento destes valores faltantes em etapas posteriores. Também veremos que os atributos já estão no intervalo entre 1 e 10, indicando que não será necessário realizar nenhum tipo de padronização nos dados. Quanto à distribuição de classes, percebemos que elas estão um pouco desbalanceadas (65%-35%), mas nada que exija neste momento algum tipo de tratamento.

```
summary(dataset) # resumo estatístico

cbind(freq=table(dataset$Class),
      percentage=prop.table(table(dataset$Class))*100) # distribu
ição de classes
```

```
> summary(dataset) # resumo estatistico
  Cl.thickness       Cell.size        Cell.shape      Marg.adhesion     Epith.c.size
 Min.   : 1.000   Min.   : 1.000   Min.   : 1.000   Min.   : 1.000   Min.   : 1.000
 1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 1.000   1st Qu.: 1.000   1st Qu.: 2.000
 Median : 4.000   Median : 1.000   Median : 2.000   Median : 1.000   Median : 2.000
 Mean   : 4.384   Mean   : 3.116   Mean   : 3.198   Mean   : 2.875   Mean   : 3.232
 3rd Qu.: 6.000   3rd Qu.: 5.000   3rd Qu.: 5.000   3rd Qu.: 4.000   3rd Qu.: 4.000
 Max.   :10.000   Max.   :10.000   Max.   :10.000   Max.   :10.000   Max.   :10.000

  Bare.nuclei      Bl.cromatin     Normal.nucleoli     Mitoses           Class
 Min.   : 1.000   Min.   : 1.000   Min.   : 1.000   Min.   : 1.000   benign   :367
 1st Qu.: 1.000   1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 1.000   malignant:193
 Median : 1.000   Median : 3.000   Median : 1.000   Median : 1.000
 Mean   : 3.468   Mean   : 3.405   Mean   : 2.877   Mean   : 1.611
 3rd Qu.: 5.000   3rd Qu.: 4.250   3rd Qu.: 4.000   3rd Qu.: 1.000
 Max.   :10.000   Max.   :10.000   Max.   :10.000   Max.   :10.000
 NA's   :13

> cbind(freq=table(dataset$Class),
+       percentage=prop.table(table(dataset$Class))*100) # distribuição de classes
          freq percentage
benign     367   65.53571
malignant  193   34.46429
```

Figura 11.3: Resultados

Exibindo a correlação entre os atributos de entrada, poderemos perceber uma alta correlação entre alguns dos atributos. Se quisermos, poderemos mais para a frente remover os atributos altamente correlacionados para aprimorar a performance dos algoritmos.

```
# Sumarização das correlações entre os atributos de entrada
complete_cases <- complete.cases(dataset)
cor(dataset[complete_cases,1:9])
```

```
> cor(dataset[complete_cases,1:9])
              Cl.thickness Cell.size Cell.shape Marg.adhesion Epith.c.size
Cl.thickness     1.0000000 0.6200884  0.6302917     0.4741733    0.5089557
Cell.size        0.6200884 1.0000000  0.9011340     0.7141150    0.7404824
Cell.shape       0.6302917 0.9011340  1.0000000     0.6846206    0.7043423
Marg.adhesion    0.4741733 0.7141150  0.6846206     1.0000000    0.5860660
Epith.c.size     0.5089557 0.7404824  0.7043423     0.5860660    1.0000000
Bare.nuclei      0.5600770 0.6687226  0.6896724     0.6660165    0.5568406
Bl.cromatin      0.5290733 0.7502700  0.7276114     0.6660533    0.6102032
Normal.nucleoli  0.5143933 0.7072182  0.7127155     0.6031036    0.6433364
Mitoses          0.3426018 0.4506532  0.4345125     0.4314910    0.4775271
              Bare.nuclei Bl.cromatin Normal.nucleoli   Mitoses
Cl.thickness    0.5600770   0.5290733       0.5143933 0.3426018
Cell.size       0.6687226   0.7502700       0.7072182 0.4506532
Cell.shape      0.6896724   0.7276114       0.7127155 0.4345125
Marg.adhesion   0.6660165   0.6660533       0.6031036 0.4314910
Epith.c.size    0.5568406   0.6102032       0.6433364 0.4775271
Bare.nuclei     1.0000000   0.6668483       0.5795794 0.3539473
Bl.cromatin     0.6668483   1.0000000       0.6838547 0.3545122
Normal.nucleoli 0.5795794   0.6838547       1.0000000 0.4084127
Mitoses         0.3539473   0.3545122       0.4084127 1.0000000
```

Figura 11.4: Resultados

Vamos agora criar algumas visualizações de dados, iniciando com as **visualizações unimodais** (histograma, gráfico de densidade e boxplots de cada atributo). Analisando os gráficos, perceberemos que quase todos os atributos seguem uma distribuição exponencial ou bimodal, podendo ser necessário transformar algumas transformações de dados mais à frente.

```
# b) Visualizações de Dados

# Visualizações unimodais

# Histogramas de cada atributo
par(mfrow=c(3,3))
for(i in 1:9) {
  hist(dataset[,i], main=names(dataset)[i])
}
```

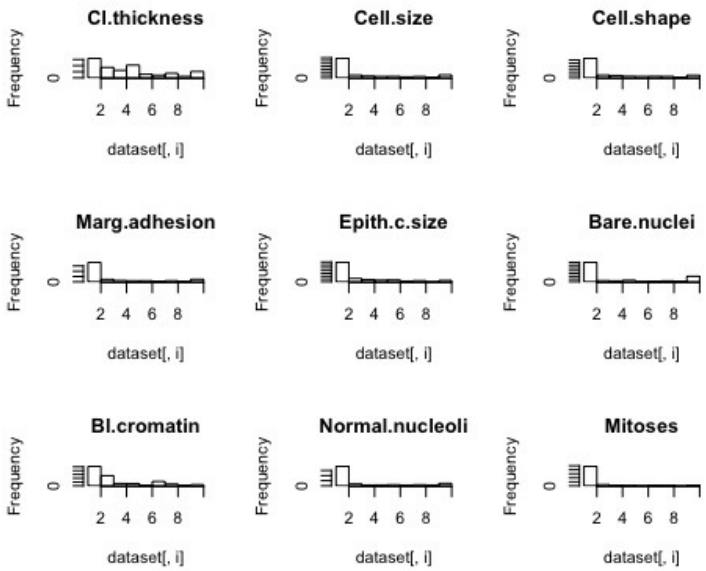



Figura 11.5: Resultados

```
# Gráficos de densidade de cada atributo
par(mfrow=c(3,3))
complete_cases <- complete.cases(dataset)
for(i in 1:9) {
  plot(density(dataset[complete_cases,i]), main=names(dataset)[i]
)
}
```

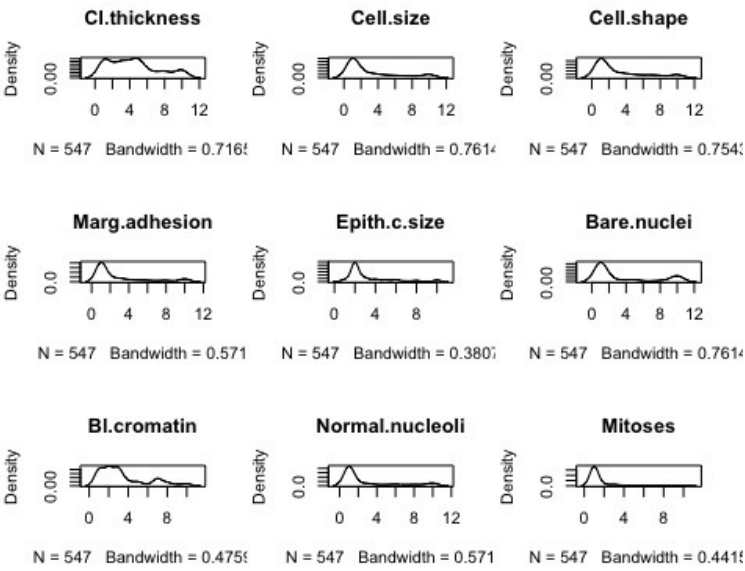



Figura 11.6: Resultados

```
# Boxplots de cada atributo
par(mfrow=c(3,3))
for(i in 1:9) {
  boxplot(dataset[,i], main=names(dataset)[i])
}
```

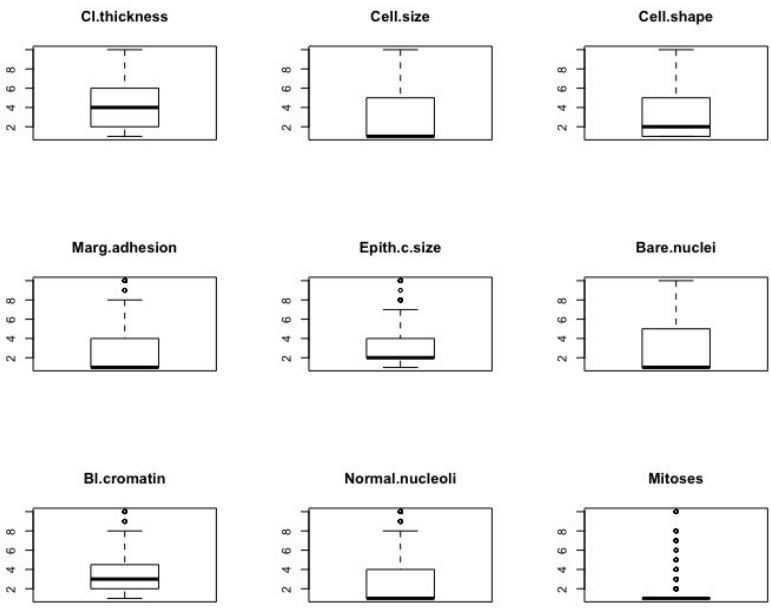



Figura 11.7: Resultados

Seguiremos com as **visualizações multimodais**, explorando as interações entre os atributos com a matriz *scatter plot* (sendo necessário adicionar algum "jitter" para facilitar a visualização dos dados) e com gráficos de barras ilustrando a relação entre os diferentes valores de cada atributo e a classe.

```
# Visualizações multimodais

# Matriz scatter plot
jittered_x <- sapply(dataset[,1:9], jitter)
pairs(jittered_x, names(dataset[,1:9]), col=dataset$Class)
```

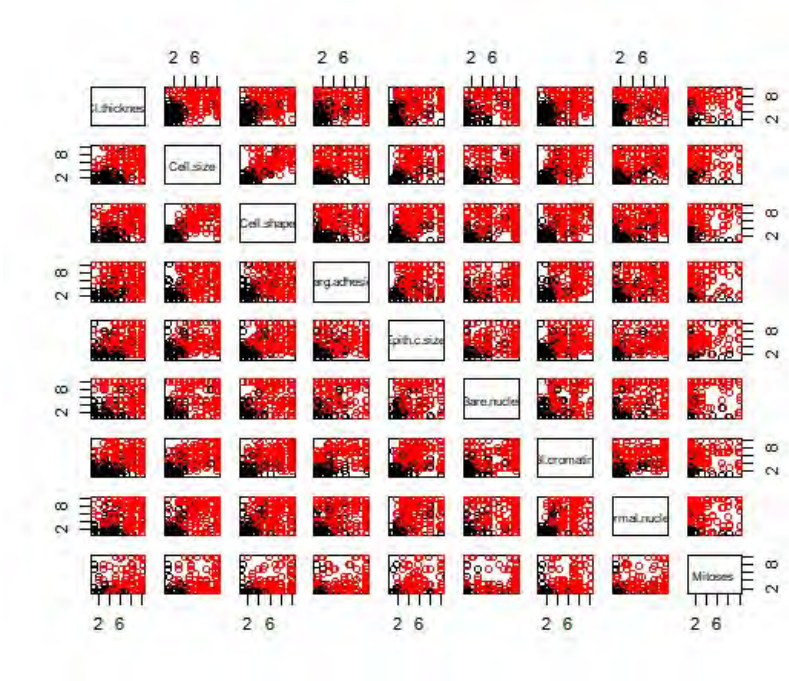



Figura 11.8: Matriz scatter plot

```
# Gráfico de barras de cada atributo por classe
par(mfrow=c(3,3))
for(i in 1:9) {
  barplot(table(dataset$Class,dataset[,i]), main=names(dataset)[i
]))
}
```

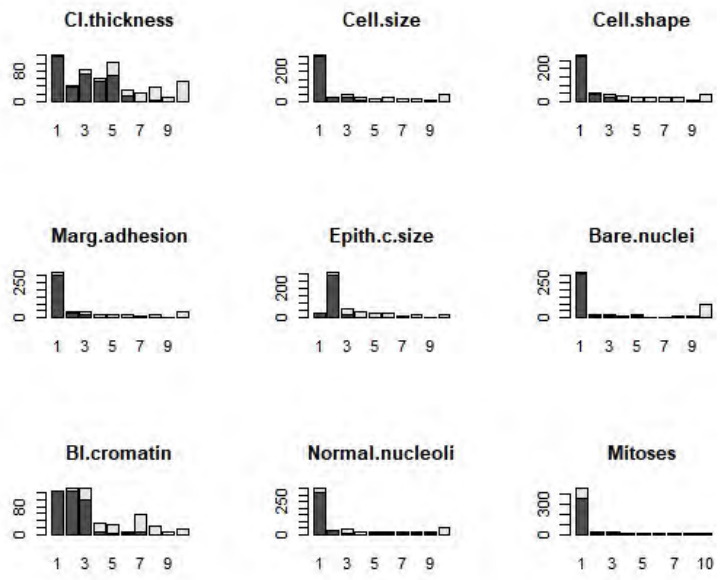



Figura 11.9: Gráfico de barras de cada atributo por classe

Seguindo o template, agora executaríamos as operações de **pré-processamento de dados**. Vamos pular esta etapa neste exemplo, uma vez que já executamos parte destas operações excluindo a coluna *Id* e convertendo os demais atributos para valores numéricos, e ainda realizaremos transformações de dados em etapas futuras. Seguiremos então para a etapa de **Modelagem e Inferência**. Definiremos, inicialmente, que utilizaremos a estratégia de treinamento com validação cruzada 10-fold e 3 repetições e a métrica de acurácia para avaliação dos resultados.

```
# 4. Modelagem e Inferência

# a) Escolha de procedimentos e métricas de avaliação
# validação cruzada 10-fold com 3 repetições
```

```
configTreino <- trainControl(method="repeatedcv", number=10, repe
ats=3)
metrica <- "Accuracy" # definição da métrica de avaliação
```

Em seguida, vamos criar 5 modelos de Classificação, usando os algoritmos **Regressão Logística**, **KNN**, **Árvore de Classificação CART**, **Naive Bayes** e **SVM**, mas sem realizar nenhum tipo de configuração específica dos hiperparâmetros dos algoritmos, utilizando a sua configuração padrão, mas indicando para que as linhas com valores NA sejam omitidas.

```
# b) Criação de Algoritmos

# Regressão Logística
modelo.glm <- train(Class~., data=dataset, method="glm", metric=m
etrica,
                    trControl=configTreino, na.action=na.omit)
# KNN
modelo.knn <- train(Class~., data=dataset, method="knn", metric=m
etrica,
                    trControl=configTreino, na.action=na.omit)

# Árvore de Classificação CART
modelo.cart <- train(Class~., data=dataset, method="rpart", metri
c=metrica,
                     trControl=configTreino, na.action=na.omit)
# Naive Bayes
modelo.nb <- train(Class~., data=dataset, method="nb", metric=met
rica, trControl=configTreino,
                   na.action=na.omit)
# SVM
modelo.svm <- train(Class~., data=dataset, method="svmRadial", me
tric=metrica,
                    trControl=configTreino, na.action=na.omit)
```

Agora vamos **comparar os resultados** dos modelos criados através da exibição dos seus valores de acurácia. Também mostraremos uma comparação gráfica dos modelos e perceberemos que acurácias mais altas são dos algoritmos de Regressão Logística e KNN.

```
# c) Comparação de Algoritmos

resultados <- resamples(list(LG=modelo.glm, KNN=modelo.knn,
                             CART=modelo.cart, NB=modelo.nb, SVM=
modelo.svm))
summary(resultados) # resultados
dotplot(resultados) # resultados gráficos
```

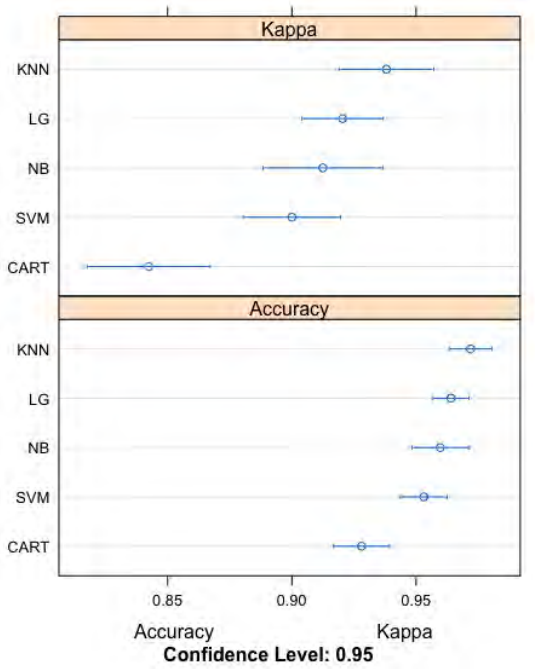



Figura 11.10: Resultados

Neste momento, vale a pena aplicarmos métodos de **transformação de dados** para ajustar e normalizar as distribuições dos atributos, e verificar se obtemos melhores resultados. Para tal, vamos utilizar a transformação *Box-Cox*, e perceberemos que os dois algoritmos que mais se destacam serão o KNN e o SVM.

```
# d) Criação de novos modelos aplicando a transformação Box-Cox n
os dados

# LG
modelo.glm <- train(Class~., data=dataset, method="glm", metric=m
etrica, preProc=c("BoxCox"),
                    trControl=configTreino, na.action=na.omit)
# KNN
modelo.knn <- train(Class~., data=dataset, method="knn", metric=m
etrica, preProc=c("BoxCox"),
                    trControl=configTreino, na.action=na.omit)
# CART
modelo.cart <- train(Class~., data=dataset, method="rpart", metri
c=metrica,
                     preProc=c("BoxCox"), trControl=configTreino,
 na.action=na.omit)
# Naive Bayes
modelo.nb <- train(Class~., data=dataset, method="nb", metric=met
rica, preProc=c("BoxCox"),
                   trControl=configTreino, na.action=na.omit)
# SVM
modelo.svm <- train(Class~., data=dataset, method="svmRadial", me
tric=metrica,
                    preProc=c("BoxCox"), trControl=configTreino,
na.action=na.omit)

# Comparação de Algoritmos
todosResultados <- resamples(list(LG=modelo.glm, KNN=modelo.knn,
                                  CART=modelo.cart, NB=modelo.nb,
 SVM=modelo.svm))
summary(todosResultados)
dotplot(todosResultados)
```

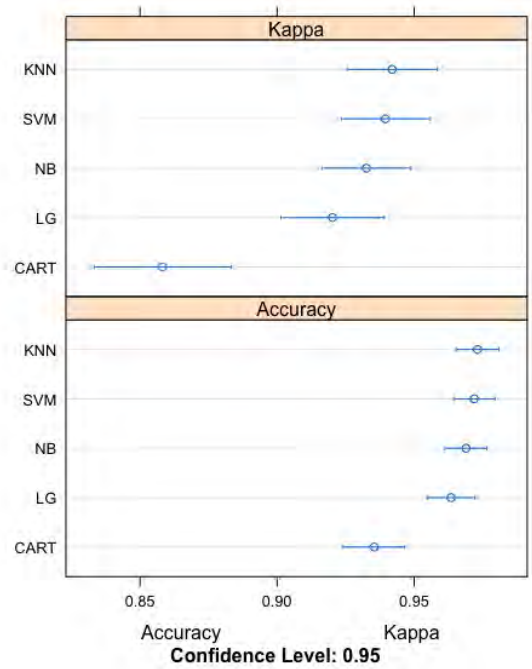



Figura 11.11: Resultados

Em seguida, vamos realizar o **ajuste dos hiperparâmetros** dos dois melhores modelos, buscando a configuração que gere os melhores resultados possíveis para cada um deles. Para tal, utilizaremos a ferramenta *grid search* do pacote *caret*, que permite que sejam definidos diversos valores para cada um dos parâmetros para cada algoritmo, e que cada combinação seja avaliada. Para o SVM, vamos variar os valores dos parâmetros sigma (parâmetro de suavização) e C (rigidez da margem) e, para o KNN, os valores do parâmetro k (número de vizinhos). Observaremos que haverá pouca variação nos resultados de acurácia de ambos os algoritmos, independente da variação de parâmetros.

```
# e) Melhoria do desempenho dos modelos através de ajuste de hipe
rparâmetros

# Tuning do SVM com Grid Search
grid <- expand.grid(.sigma=c(0.025, 0.05, 0.1, 0.15), .C=seq(1, 1(
, by=1))
modelo.svm <- train(Class~., data=dataset, method="svmRadial", me
tric=metrica, tuneGrid=grid,
                    preProc=c("BoxCox"), trControl=configTreino,
na.action=na.omit)
print(modelo.svm) # resultados
plot(modelo.svm) # resultados gráficos

# Tuning do KNN com Grid Search
grid <- expand.grid(.k=seq(1,20,by=1))
modelo.knn <- train(Class~., data=dataset, method="knn", metric=m
etrica, tuneGrid=grid,
                    preProc=c("BoxCox"), trControl=configTreino,
na.action=na.omit)
print(modelo.knn) # resultados
plot(modelo.knn) # resultados gráficos
```

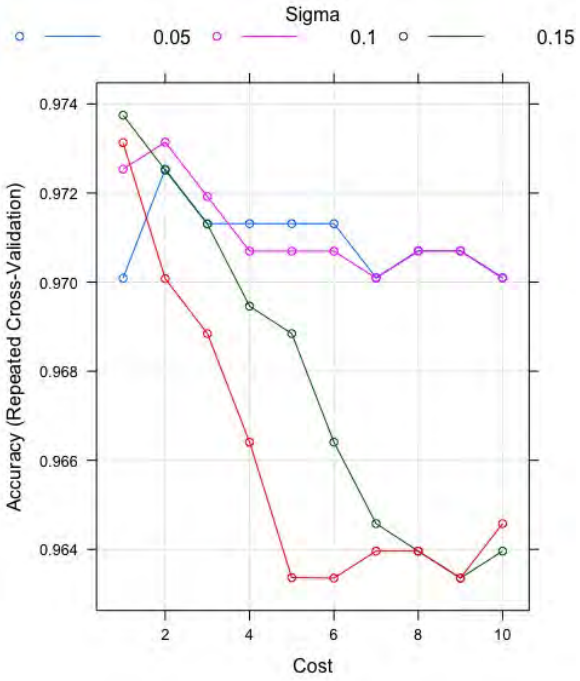



Figura 11.12: Resultados

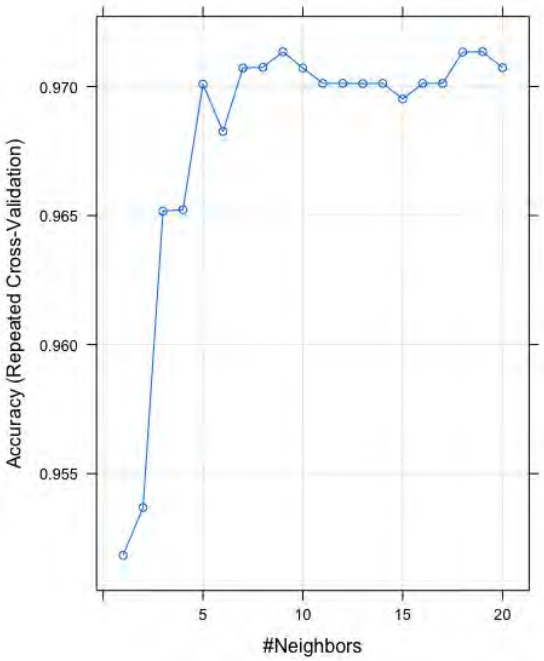



Figura 11.13: Resultados

Uma vez que os resultados foram bem similares, escolheremos o KNN como modelo final, por exigir menos memória para execução em comparação com o SVM. Entramos, então, na etapa de **pós-processamento**, na qual vamos construir o modelo final escolhido utilizando todos os dados de treinamento, e preparar o conjunto de testes para a validação final do modelo, simulando a sua execução em dados ainda não vistos pelo modelo. Será necessário realizar todas as operações de pré-processamento de dados que realizamos nos dados de treinamento também neste conjunto (remoção do atributo *Id*, remoção das linhas com valores faltantes, conversão do tipo dos atributos para numérico, aplicação

da transformação *Box-Cox*).

```
# 5. Pós-processamento

# a) Escolha e construção do modelo final com todo o conjunto de
treinamento

# Transformação de dados
datasetSemMissing <- dataset[complete.cases(dataset),]
x <- datasetSemMissing[,1:9]
parametrosPreProcessamento <- preProcess(x, method=c("BoxCox"))
x <- predict(parametrosPreProcessamento, x)

# Preparação do conjunto de teste (para validação final)
conjTeste <- conjTeste[,-1] # remoção do atributo Id
conjTeste <- conjTeste[complete.cases(conjTeste),] # remoção dos
valores faltantes

for(i in 1:9) { # conversão dos valores de entrada para numéricos
  conjTeste[,i] <- as.numeric(as.character(conjTeste[,i]))
}
conjTesteX <- predict(parametrosPreProcessamento, conjTeste[,1:9]
) # transformação de dados

# b) Predições no conjunto de teste (validação final)
predicoes <- knn3Train(x, conjTesteX, datasetSemMissing$Class, k=
, prob=FALSE) # predições
confusionMatrix(as.factor(predicoes), conjTeste$Class) # matriz d
e confusão
```

```
> confusionMatrix(as.factor(predicoes), conjTeste$Class) # matriz de confusão
Confusion Matrix and Statistics

          Reference
Prediction  benign malignant
  benign        87         0
  malignant      1        48

               Accuracy : 0.9926
                 95% CI : (0.9597, 0.9998)
    No Information Rate : 0.6471
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.984
 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9886
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.9796
             Prevalence : 0.6471
         Detection Rate : 0.6397
   Detection Prevalence : 0.6397
      Balanced Accuracy : 0.9943

       'Positive' Class : benign
```

Figura 11.14: Pós-processamento

Analisando o resultado do modelo nas predições do conjunto de testes (dados que até o momento não tinham sido vistos pelo modelo, uma vez que não foram utilizados no seu treinamento), percebemos que alcançamos uma boa acurácia, indicando que este modelo provavelmente apresentará bons resultados preditivos em dados não vistos. A próxima etapa consistirá em **salvar** esta versão final do modelo, para que seja possível carregá-lo para realizar predições quando novos dados estiverem disponíveis:

```
# c) Salvamento do modelo para uso posterior

# Salvamento do modelo
modeloFinal <- modelo.knn
saveRDS(modeloFinal, "./modeloFinal.rds")
```

```
#... no futuro, poderemos carregar o modelo
modeloCarregado <- readRDS("./modeloFinal.rds")
```

## 11.3 E AGORA?

Chegamos ao final deste livro com o objetivo de explorar o processo de Ciência de Dados cumprido, mas este é apenas o começo dos seus estudos na área de ciência de dados, que contempla uma infinidade de técnicas, modelos e aplicações interessantes. Estamos compilando em https://tinyurl.com/usxuuhf uma lista de indicações de livros, blogs, cursos e outros materiais relacionados a Ciência de Dados, que está em constante atualização e não tem o propósito de ser exaustiva, uma vez que novos materiais interessantes surgem todos os dias! Caso você tenha sugestões de outros conteúdos, sinta-se à vontade para deixar um comentário no artigo com sua indicação.